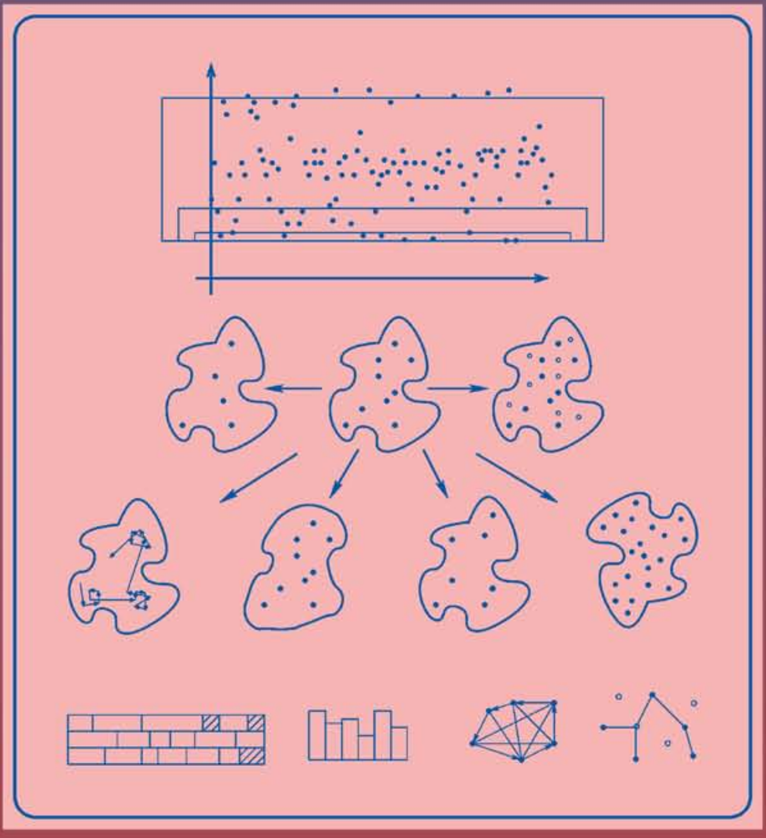
CHAPMAN & HALL/CRC COMPUTER and INFORMATION SCIENCE SERIES

Handbook of Approximation Algorithms and Metaheuristics



Edited by

Teofilo F. Gonzalez

Chapman & Hall/CRC
Taylor & Francis Group

Handbook of Approximation Algorithms and Metaheuristics

CHAPMAN & HALL/CRC COMPUTER and INFORMATION SCIENCE SERIES

Series Editor: Sartaj Sahni

PUBLISHED TITLES

ADVERSARIAL REASONING: COMPUTATIONAL APPROACHES TO READING THE OPPONENT'S MIND
Alexander Kott and William M. McEneaney

DISTRIBUTED SENSOR NETWORKS
S. Sitharama Iyengar and Richard R. Brooks

DISTRIBUTED SYSTEMS: AN ALGORITHMIC APPROACH
Sukumar Ghosh

FUNDAMENTALS OF NATURAL COMPUTING: BASIC CONCEPTS, ALGORITHMS, AND APPLICATIONS
Leandro Nunes de Castro

HANDBOOK OF ALGORITHMS FOR WIRELESS NETWORKING AND MOBILE COMPUTING
Azzedine Boukerche

HANDBOOK OF APPROXIMATION ALGORITHMS AND METAHEURISTICS
Teofilo F. Gonzalez

HANDBOOK OF BIOINSPIRED ALGORITHMS AND APPLICATIONS
Stephan Olariu and Albert Y. Zomaya

HANDBOOK OF COMPUTATIONAL MOLECULAR BIOLOGY
Srinivas Aluru

HANDBOOK OF DATA STRUCTURES AND APPLICATIONS
Dinesh P. Mehta and Sartaj Sahni

HANDBOOK OF SCHEDULING: ALGORITHMS, MODELS, AND PERFORMANCE ANALYSIS
Joseph Y.-T. Leung

THE PRACTICAL HANDBOOK OF INTERNET COMPUTING
Munindar P. Singh

SCALABLE AND SECURE INTERNET SERVICES AND ARCHITECTURE
Cheng-Zhong Xu

SPECULATIVE EXECUTION IN HIGH PERFORMANCE COMPUTER ARCHITECTURES
David Kaeli and Pen-Chung Yew

Handbook of Approximation Algorithms and Metaheuristics

Edited by
Teofilo F. Gonzalez
University of California
Santa Barbara, U.S.A.

Chapman & Hall/CRC
Taylor & Francis Group
Boca Raton London New York

Chapman & Hall/CRC is an imprint of the
Taylor & Francis Group, an **informa** business

Chapman & Hall/CRC
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-10: 1-58488-550-5 (Hardcover)
International Standard Book Number-13: 978-1-58488-550-4 (Hardcover)

Library of Congress Cataloging-in-Publication Data

Handbook of approximation algorithms and metaheurististics / edited by Teofilo F. Gonzalez.
p. cm. -- (Chapman & Hall/CRC computer & information science ; 10)
Includes bibliographical references and index.
ISBN-13: 978-1-58488-550-4
ISBN-10: 1-58488-550-5
1. Computer algorithms. 2. Mathematical optimization. I. Gonzalez, Teofilo F. II. Title. III. Series.

QA76.9.A43H36 2007
005.1--dc22 2007002478

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

DEDICATED

To my wife
Dorothy,

and my children
Jeanmarie, Alexis, Julia, Teofilo, and Paolo.

Preface

Forty years ago (1966), Ronald L. Graham formally introduced approximation algorithms. The idea was to generate near-optimal solutions to optimization problems that could not be solved efficiently by the computational techniques available at that time. With the advent of the theory of NP-completeness in the early 1970s, the area became more prominent as the need to generate near optimal solutions for NP-hard optimization problems became the most important avenue for dealing with computational intractability. As it was established in the 1970s, for some problems one can generate near optimal solutions quickly, while for other problems generating provably good suboptimal solutions is as difficult as generating optimal ones. Other approaches based on probabilistic analysis and randomized algorithms became popular in the 1980s. The introduction of new techniques to solve linear programming problems started a new wave for developing approximation algorithms that matured and saw tremendous growth in the 1990s. To deal, in a practical sense, with the inapproximable problems there were a few techniques introduced in the 1980s and 1990s. These methodologies have been referred to as metaheuristics. There has been a tremendous amount of research in metaheuristics during the past two decades. During the last 15 or so years approximation algorithms have attracted considerably more attention. This was a result of a stronger inapproximability methodology that could be applied to a wider range of problems and the development of new approximation algorithms for problems in traditional and emerging application areas.

As we have witnessed, there has been tremendous growth in field of approximation algorithms and metaheuristics. The basic methodologies are presented in Parts I–III. Specifically, Part I covers the basic methodologies to design and analyze efficient approximation algorithms for a large class of problems, and to establish inapproximability results for another class of problems. Part II discusses local search, neural networks and metaheuristics. In Part III multiobjective problems, sensitivity analysis and stability are discussed.

Parts IV–VI discuss the application of the methodologies to classical problems in combinatorial optimization, computational geometry and graphs problems, as well as for large-scale and emerging applications. The approximation algorithms discussed in the handbook have primary applications in computer science, operations research, computer engineering, applied mathematics, bioinformatics, as well as in engineering, geography, economics, and other research areas with a quantitative analysis component.

Chapters 1 and 2 present an overview of the field and the handbook. These chapters also cover basic definitions and notation, as well as an introduction to the basic methodologies and inapproximability. Chapters 1–8 discuss methodologies to develop approximation algorithms for a large class of problems. These methodologies include restriction (of the solution space), greedy methods, relaxation (LP and SDP) and rounding (deterministic and randomized), and primal-dual methods. For a minimization problem P these methodologies provide for every problem instance I a solution with objective function value that is at most $(1+\epsilon) \cdot f^*(I)$, where ϵ is a positive constant (or a function that depends on the instance size) and $f^*(I)$ is the optimal solution value for instance I. These algorithms take polynomial time with respect to the size of the instance I being solved. These techniques also apply to maximization problems, but the guarantees are different. Given as input a value for ϵ and any instance I for a given problem P, an approximation scheme finds a solution with objective function value at most $(1+\epsilon) \cdot f^*(I)$. Chapter 9 discusses techniques that have been used to design approximation schemes. These approximation schemes take polynomial time with respect to the size of the instance I (PTAS). Chapter 10 discusses different methodologies for designing fully polynomial approximation schemes (FPTAS). These schemes take polynomial time with respect to the size of the instance I and $1/\epsilon$. Chapters 11–13 discuss asymptotic and randomized approximation schemes, as well as distributed and randomized approximation algorithms. Empirical analysis is covered in Chapter 14 as well as in chapters in Parts IV–VI. Chapters 15–17 discuss performance measures, reductions that preserve approximability, and inapproximability results.

Part II discusses deterministic and stochastic local search as well as very large neighborhood search. Chapters 21 and 22 present reactive search and neural networks. Tabu search, evolutionary computation, simulated annealing, ant colony optimization and memetic algorithms are covered in Chapters 23–27. In Part III, I discuss multiobjective optimization problems, sensitivity analysis and stability of approximations.

Part IV covers traditional applications. These applications include bin packing and extensions, packing problems, facility location and dispersion, traveling salesperson and generalizations, Steiner trees, scheduling, planning, generalized assignment, and satisfiability.

Computational geometry and graph applications are discussed in Part V. The problems discussed in this part include triangulations, connectivity problems in geometric graphs and networks, dilation and detours, pair decompositions, partitioning (points, grids, graphs and hypergraphs), maximum planar subgraphs, edge disjoint paths and unsplittable flow, connectivity problems, communication spanning trees, most vital edges, and metaheuristics for coloring and maximum disjoint paths.

Large-scale and emerging applications (Part VI) include chapters on wireless ad hoc networks, sensor networks, topology inference, multicast congestion, QoS multimedia routing, peer-to-peer networks, data broadcasting, bioinformatics, CAD and VLSI applications, game theoretic approximation, approximating data streams, digital reputation and color quantization.

Readers who are not familiar with approximation algorithms and metaheuristics should begin with Chapters 1–6, 9–10, 18–21, and 23–27. Experienced researchers will also find useful material in these basic chapters. We have collected in this volume a large amount of this material with the goal of making it as complete as possible. I apologize in advance for omissions and would like to invite all of you to suggest to me chapters (for future editions of this handbook) to keep up with future developments in the area. I am confident that research in the field of approximations algorithms and metaheuristics will continue to flourish for a few more decades.

Teofilo F. Gonzalez
Santa Barbara, California

About the Cover

The four objects in the bottom part of the cover represent scheduling, bin packing, traveling salesperson, and Steiner tree problems. A large number of approximation algorithms and metaheuristics have been designed for these four fundamental problems and their generalizations.

The seven objects in the middle portion of the cover represent the basic methodologies. Of these seven, the object in the top center represents a problem by its solution space. The object to its left represents its solution via restriction and the one to its right represents relaxation techniques. The objects in the row below represent local search and metaheuristics, problem transformation, rounding, and primal-dual methods.

The points in the top portion of the cover represent solutions to a problem and their height represents their objective function value. For a minimization problem, the possible solutions generated by an approximation scheme are the ones inside the bottommost rectangle. The ones inside the next rectangle represent the one generated by a constant ratio approximation algorithm. The top rectangle represents the possible solution generated by a polynomial time algorithm for inapproximable problems (under some complexity theoretic hypothesis).

About the Editor

Dr. Teofilo F. Gonzalez received the B. S. degree in computer science from the Instituto Tecnológico de Monterrey (1972). He was one of the first handful of students to receive a computer science degree in Mexico. He received his Ph.D. degree from the University of Minnesota, Minneapolis (1975). He has been member of the faculty at Oklahoma University, Penn State, and University of Texas at Dallas, and has spent sabbatical leaves at Utrecht University (Netherlands) and the Instituto Tecnológico de Monterrey (ITESM, Mexico). Currently he is professor of computer science at the University of California, Santa Barbara. Professor Gonzalez's main area of research activity is the design and analysis of efficient exact and approximation algorithms for fundamental problems arising in several disciplines. His main research contributions fall in the areas of resource allocation and job scheduling, message dissemination in parallel and distributed computing, computational geometry, graph theory, and VLSI placement and wire routing.

His professional activities include chairing conference program committees and membership in journal editorial boards. He has served as an accreditation evaluator and has been a reviewer for numerous journals and conferences, as well as CS programs and funding agencies.

Contributors

Emile Aarts
Philips Research Laboratories
Eindhoven, The Netherlands

Ravindra K. Ahuja
University of Florida
Gainesville, Florida

Enrique Alba
University of Málaga
Málaga, Spain

Christoph Albrecht
Cadence Berkeley Labs
Berkeley, California

Eric Angel
University of Evry Val d'Essonne
Evry, France

Abdullah N. Arslan
University of Vermont
Burlington, Vermont

Giorgio Ausiello
University of Rome "La Sapienza"
Rome, Italy

Sudha Balla
University of Connecticut
Storrs, Connecticut

Evripidis Bampis
University of Evry Val d'Essonne
Evry, France

Roberto Battiti
University of Trento
Trento, Italy

Alan A. Bertossi
University of Bologna
Bologna, Italy

Maria J. Blesa
Technical University of Catalonia
Barcelona, Spain

Christian Blum
Technical University of Catalonia
Barcelona, Spain

Vincenzo Bonifaci
University of Rome "La Sapienza"
Rome, Italy

Hans-Joachim Böckenhauer
Swiss Federal Institute of Technology (ETH) Zürich
Zürich, Switzerland

Mauro Brunato
University of Trento
Povo, Italy

Gruia Calinescu
Illinois Institute of Technology
Chicago, Illinois

Peter Cappello
University of California, Santa Barbara
Santa Barbara, California

Kun-Mao Chao
National Taiwan University
Taiwan, Republic of China

Danny Z. Chen
University of Notre Dame
Notre Dame, Indiana

Ting Chen
University of Southern California
Los Angeles, California

Marco Chiarandini
University of Southern Denmark
Odense, Denmark

Francis Y. L. Chin
The University of Hong Kong
Hong Kong, China

Christopher James Coakley
University of California, Santa Barbara
Santa Barbara, California

Edward G. Coffman, Jr.
Columbia University
New York, New York

Jason Cong
University of California
Los Angeles, California

Carlos Cotta
University of Málaga
Málaga, Spain

János Csirik
University of Szeged
Szeged, Hungary

Artur Czumaj
University of Warwick
Coventry, United Kingdom

Bhaskar DasGupta
University of Illinois at Chicago
Chicago, Illinois

Jaime Davila
University of Connecticut
Storrs, Connecticut

Xiaotie Deng
City University of Hong Kong
Hong Kong, China

Marco Dorigo
Free University of Brussels
Brussels, Belgium

Ding-Zhu Du
University of Texas at Dallas
Richardson, Texas

Devdatt Dubhashi
Chalmers University
Goteborg, Sweden

Irina Dumitrescu
HEC Montréal
Montreal, Canada, and
University of New South Wales
Sydney, Australia

Ömer Eğecioğlu
University of California, Santa Barbara
Santa Barbara, California

Leah Epstein
University of Haifa
Haifa, Israel

Özlem Ergun
Georgia Institute of Technology
Atlanta, Georgia

Guy Even
Tel Aviv University
Tel Aviv, Israel

Cristina G. Fernandes
University of São Paulo
São Paulo, Brazil

David Fernández-Baca
Iowa State University
Ames, Iowa

Jeremy Frank
NASA Ames Research Center
Moffett Field, California

Stanley P. Y. Fung
University of Leicester
Leicester, United Kingdom

Anurag Garg
University of Trento
Trento, Italy

Daya Ram Gaur
University of Lethbridge
Lethbridge, Canada

Silvia Ghilezan
University of Novi Sad
Novi Sad, Serbia

Fred Glover
University of Colorado
Boulder, Colorado

Teofilo F. Gonzalez
University of California, Santa Barbara
Santa Barbara, California

Laurent Gourvès
University of Evry Val d'Essonne
Evry, France

Fabrizio Grandoni
University of Rome "La Sapienza"
Rome, Italy

Joachim Gudmundsson
National ICT Australia Ltd
Sydney, Australia

Sudipto Guha
University of Pennsylvania
Philadelphia, Pennsylvania

Hann-Jang Ho
Wufeng Institute of Technology
Taiwan, Republic of China

Holger H. Hoos
University of British Columbia
Vancouver, Canada

Juraj Hromkovič
Swiss Federal Institute of Technology (ETH) Zürich
Zürich, Switzerland

Li-Sha Huang
Tsinghua University
Beijing, China

Yao-Ting Huang
National Taiwan University
Taiwan, Republic of China

Toshihide Ibaraki
Kwansei Gakuin University
Sanda, Japan

Shinji Imahori
University of Tokyo
Tokyo, Japan

Klaus Jansen
Kiel University
Kiel, Germany

Ari Jónsson
NASA Ames Research Center
Moffett Field, California

Andrew B. Kahng
University of California at San Diego
La Jolla, California

Yoshiyuki Karuno
Kyoto Institute of Technology
Kyoto, Japan

Samir Khuller
University of Maryland
College Park, Maryland

Christian Knauer
Free University of Berlin
Berlin, Germany

Rajeev Kohli
Columbia University
New York, New York

Stavros G. Kolliopoulos
National and Kapodistrian University of Athens
Athens, Greece

Jan Korst
Philips Research Laboratories
Eindhoven, The Netherlands

Guy Kortsarz
Rutgers University
Camden, New Jersey

Sofia Kovaleva
University of Maastricht
Maastricht, The Netherlands

Ramesh Krishnamurti
Simon Fraser University
Burnaby, Canada

Manuel Laguna
University of Colorado
Boulder, Colorado

Michael A. Langston
University of Tennessee
Knoxville, Tennessee

Sing-Ling Lee
National Chung-Cheng University
Taiwan, Republic of China

Guillermo Leguizamón
National University of San Luis
San Luis, Argentina

Stefano Leonardi
University of Rome "La Sapienza"
Rome, Italy

Joseph Y.-T. Leung
New Jersey Institute of Technology
Newark, New Jersey

Xiang-Yang Li
Illinois Institute of Technology
Chicago, Illinois

Andrzej Lingas
Lund University
Lund, Sweden

Derong Liu
University of Illinois at Chicago
Chicago, Illinois

Errol L. Lloyd
University of Delaware
Newark, Delaware

Ion Măndoiu
University of Connecticut
Storrs, Connecticut

Alberto Marchetti-Spaccamela
University of Rome "La Sapienza"
Rome, Italy

Igor L. Markov
University of Michigan
Ann Arbor, Michigan

Rafael Martí
University of Valencia
Valencia, Spain

Wil Michiels
Philips Research Laboratories
Eindhoven, The Netherlands

Burkhard Monien
University of Paderborn
Paderborn, Germany

Pablo Moscato
The University of Newcastle
Callaghan, Australia

Rajeev Motwani
Stanford University
Stanford, California

Hiroshi Nagamochi
Kyoto University
Kyoto, Japan

Sotiris Nikoletseas
University of Patras and CTI
Patras, Greece

Zeev Nutov
The Open University of Israel
Raanana, Israel

Liadan O'Callaghan
Google
Mountain View, California

Stephan Olariu
Old Dominion University
Norfolk, Virginia

Alex Olshevsky
Massachusetts Institute of Technology
Cambridge, Massachusetts

James B. Orlin
Massachusetts Institute of Technology
Cambridge, Massachusetts

Alessandro Panconesi
University of Rome "La Sapienza"
Rome, Italy

Jovanka Pantović
University of Novi Sad
Novi Sad, Serbia

David A. Papa
University of Michigan
Ann Arbor, Michigan

Luís Paquete
University of the Algarve
Faro, Portugal

Vangelis Th. Paschos
LAMSADE CNRS UMR 7024 and University of Paris–Dauphine
Paris, France

Fanny Pascual
University of Evry Val d'Essonne
Evry, France

M. Cristina Pinotti
University of Perugia
Perugia, Italy

Robert Preis
University of Paderborn
Paderborn, Germany

Abraham P. Punnen
Simon Fraser University
Surrey, Canada

Yuval Rabani
Technion—Israel Institute of Technology
Haifa, Israel

Balaji Raghavachari
University of Texas at Dallas
Richardson, Texas

Sanguthevar Rajasekaran
University of Connecticut
Storrs, Connecticut

S. S. Ravi
University at Albany—State University of New York
Albany, New York

Andréa W. Richa
Arizona State University
Tempe, Arizona

Romeo Rizzi
University of Udine
Udine, Italy

Daniel J. Rosenkrantz
University at Albany—State University of New York
Albany, New York

Pedro M. Ruiz
University of Murcia
Murcia, Spain

Sartaj Sahni
University of Florida
Gainesville, Florida

Stefan Schamberger
University of Paderborn
Paderborn, Germany

Christian Scheideler
Technical University of Munich
Garching, Germany

Sebastian Seibert
Swiss Federal Institute of Technology (ETH) Zürich
Zürich, Switzerland

Hadas Shachnai
Technion—Israel Institute of Technology
Haifa, Israel

Hong Shen
University of Adelaide
Adelaide, Australia

Joseph R. Shinnerl
Tabula, Inc.
Santa Clara, California

Hava T. Siegelmann
University of Massachusetts
Amherst, Massachusetts

Michiel Smid
Carleton University
Ottawa, Canada

Anthony Man-Cho So
Stanford University
Stanford, California

Krzysztof Socha
Free University of Brussels
Brussels, Belgium

Roberto Solis-Oba
The University of Western Ontario
London, Canada

Frits C. R. Spieksma
Catholic University of Leuven
Leuven, Belgium

Paul Spirakis
University of Patras and CTI
Patras, Greece

Rob van Stee
University of Karlsruhe
Karlsruhe, Germany

Ivan Stojmenovic
University of Ottawa
Ottawa, Canada

Thomas Stützle
Free University of Brussels
Brussels, Belgium

Mario Szegedy
Rutgers University
Piscataway, New Jersey

Chuan Yi Tang
National Tsing Hua University
Taiwan, Republic of China

Giri K. Tayi
University at Albany—State University of New York
Albany, New York

Tami Tamir
The Interdisciplinary Center
Herzliya, Israel

Hui Tian
University of Science and Technology of China
Hefei, China

Balaji Venkatachalam
University of California, Davis
Davis, California

Cao-An Wang
Memorial University of Newfoundland
St. John's, Newfoundland, Canada

Lan Wang
Old Dominion University
Norfolk, Virginia

Yu Wang
University of North Carolina at Charlotte
Charlotte, North Carolina

Weizhao Wang
Illinois Institute of Technology
Chicago, Illinois

Bang Ye Wu
Shu-Te University
Taiwan, Republic of China

Weili Wu
University of Texas at Dallas
Richardson, Texas

Zhigang Xiang
Queens College of the City University of New York
Flushing, New York

Jinhui Xu
State University of New York at Buffalo
Buffalo, New York

Mutsunori Yagiura
Nagoya University
Nagoya, Japan

Rong-Jou Yang
Wufeng Institute of Technology
Taiwan, Republic of China

Yinyu Ye
Stanford University
Stanford, California

Neal E. Young
University of California at Riverside
Riverside, California

Alexander Zelikovsky
Georgia State University
Atlanta, Georgia

Hu Zhang
McMaster University
Hamilton, Canada

Jiawei Zhang
New York University
New York, New York

Kui Zhang
University of Alabama at Birmingham
Birmingham, Alabama

Si Qing Zheng
University of Texas at Dallas
Richardson, Texas

An Zhu
Google
Mountain View, California

Joviša Žunić
University of Exeter
Exeter, United Kingdom

Contents

PART I Basic Methodologies

PART II Local Search, Neural Networks, and Metaheuristics

PART III Multiobjective Optimization, Sensitivity Analysis, and Stability

PART IV Traditional Applications

PART V Computational Geometry and Graph Applications

PART VI Large-Scale and Emerging Applications

I

Basic Methodologies

1

Introduction, Overview, and Notation

Teofilo F. Gonzalez
University of California, Santa Barbara

1.1 Introduction

Approximation algorithms, as we know them now, were formally introduced in the 1960s to generate near-optimal solutions to optimization problems that could not be solved efficiently by the computational techniques available at that time. With the advent of the theory of NP-completeness in the early 1970s, the area became more prominent as the need to generate near-optimal solutions for NP-hard optimization problems became the most important avenue for dealing with computational intractability. As established in the 1970s, for some problems one can generate near-optimal solutions quickly, while for other problems generating provably good suboptimal solutions is as difficult as generating optimal ones. Other approaches based on probabilistic analysis and randomized algorithms became popular in the 1980s. The introduction of new techniques to solve linear programming problems started a new wave for developing approximation algorithms that matured and saw tremendous growth in the 1990s. To deal, in a practical sense, with the inapproximable problems, there were a few techniques introduced in the 1980s and 1990s. These methodologies have been referred to as metaheuristics and include simulated annealing (SA), ant colony optimization (ACO), evolutionary computation (EC), tabu search (TS), and memetic algorithms (MA). Other previously established methodologies such as local search, backtracking, and branch-and-bound were also explored at that time. There has been a tremendous amount of research in metaheuristics during the past two decades. These techniques have been evaluated experimentally and have demonstrated their usefulness for solving practical problems. During the past 15 years or so, approximation algorithms have attracted considerably more attention. This was a result of a stronger inapproximability methodology that could be applied to a wider range of problems and the development of new approximation algorithms for problems arising in established and emerging application areas. Polynomial time approximation schemes (PTAS) were introduced in the 1960s and the more powerful fully polynomial time approximation schemes (FPTAS) were introduced in the 1970s. Asymptotic PTAS and FPTAS, and fully randomized approximation schemes were introduced later on.

Today, approximation algorithms enjoy a stature comparable to that of algorithms in general and the area of metaheuristics has established itself as an important research area. The new stature is a by-product of a natural expansion of research into more practical areas where solutions to real-world problems are expected, as well as by the higher level of sophistication required to design and analyze these new procedures. The goal of approximation algorithms and metaheuristics is to provide the best possible solutions and to guarantee that such solutions satisfy certain important properties. This volume houses these two approaches and thus covers all the aspects of approximations. We hope it will serve as a valuable reference for approximation methodologies and applications.

Approximation algorithms and metaheuristics have been developed to solve a wide variety of problems. A good portion of these results have only theoretical value due to the fact that their time complexity is a high-order polynomial or have huge constants associated with their time complexity bounds. However, these results are important because they establish what is possible, and it may be that in the near future these algorithms will be transformed into practical ones. Other approximation algorithms do not suffer from this pitfall, but some were designed for problems with limited applicability. However, the remaining approximation algorithms have real-world applications. Given this, there is a huge number of important application areas, including new emerging ones, where approximation algorithms and metaheuristics have barely penetrated and where we believe there is an enormous potential for their use. Our goal is to collect a wide portion of the approximation algorithms and metaheuristics in as many areas as possible, as well as to introduce and explain in detail the different methodologies used to design these algorithms.

1.2 Overview

Our overview in this section is devoted mainly to the earlier years. The individual chapters discuss in detail recent research accomplishments in different subareas. This section will also serve as an overview of Parts I, II, and III of this handbook. Chapter 2 discusses some of the basic methodologies and applies them to simple problems. This prepares the reader for the overview of Parts IV, V, and VI presented in Chapter 2.

Even before the 1960s, research in applied mathematics and graph theory had established upper and lower bounds for certain properties of graphs. For example, bounds had been established for the chromatic number, achromatic number, chromatic index, maximum clique, maximum independent set, etc. Some of these results could be seen as the precursors of approximation algorithms. By the 1960s, it was understood that there were problems that could be solved efficiently, whereas for other problems all their known algorithms required exponential time. Heuristics were being developed to find quick solutions to problems that appeared to be computationally difficult to solve. Researchers were experimenting with heuristics, branch-and-bound procedures, and iterative improvement frameworks and were evaluating their performance when solving actual problem instances. There were many claims being made, not all of which could be substantiated, about the performance of the procedures being developed to generate optimal and suboptimal solutions to combinatorial optimization problems.

1.2.1 Approximation Algorithms

Forty years ago (1966), Ronald L. Graham [1] formally introduced approximation algorithms. He analyzed the performance of list schedules for scheduling tasks on identical machines, a fundamental problem in scheduling theory.

Problem: Scheduling tasks on identical machines.

Instance: Set of n tasks $(T_1, T_2, \ldots, T_n)$ with processing time requirements $t_1, t_2, \ldots, t_n$, partial order C defined over the set of tasks to enforce task dependencies, and a set of m identical machines.

Objective: Construct a schedule with minimum makespan. A *schedule* is an assignment of tasks to time intervals on the machines in such a way that (1) each task T_i is processed continuously for t_i units of time by one of the machines; (2) each machine processes at most one task at a time; and (3) the precedence constraints are satisfied (i.e., machines cannot commence the processing of a task until all its predecessors have been completed). The *makespan* of a schedule is the time at which all the machines have completed processing the tasks.

The *list scheduling* procedure is given an ordering of the tasks specified by a list L. The procedure finds the earliest time t when a machine is idle and an unassigned task is *available* (i.e., all its predecessors have been completed). It assigns the leftmost available task in the list L to an idle machine at time t and this step is repeated until all the tasks have been scheduled.

The main result in Ref. [1] is proving that for every problem instance I, the schedule generated by this policy has a makespan that is bounded above by $(2 - 1/m)$ times the optimal makespan for the instance. This is called the *approximation ratio* or *approximation factor* for the algorithm. We also say that the algorithm is a $(2 - 1/m)$-approximation algorithm. This criterion for measuring the quality of the solutions generated by an algorithm remains one of the most important ones in use today. The second contribution in Ref. [1] is establishing that the approximation ratio $(2 - 1/m)$ is the best possible for list schedules, i.e., the analysis of the approximation ratio for this algorithm cannot be improved. This was established by presenting problem instances (for all m and $n \geq 2m - 1$) and lists for which the schedule generated by the procedure has a makespan equal to $2 - 1/m$ times the optimal makespan for the instance. A restricted version of the list scheduling algorithm is analyzed in detail in Chapter 2.

The third important result in Ref. [1] is showing that list scheduling procedures schedules may have anomalies. To explain this, we need to define some terms. The makespan of the list schedule, for instance, I, using list L is denoted by $f_L(I)$. Suppose that instance I' is a slightly modified version of instance I. The modification is such that we intuitively expect that $f_L(I') \leq f_L(I)$. But that is not always true, so there is an *anomaly*. For example, suppose that I' is I, except that I' has an additional machine. Intuitively, $f_L(I') \leq f_L(I)$ because with one additional machine tasks should be completed earlier or at the same time as when there is one fewer machine. But this is not always the case for list schedules, there are problem instances and lists for which $f_L(I') > f_L(I)$. This is called an *anomaly*. Our expectation would be valid if list scheduling would generate minimum makespan schedules. But we have a procedure that generates suboptimal solutions. Such guarantees are not always possible in this environment. List schedules suffer from other anomalies. For example, relaxing the precedence constraints or decreasing the execution time of the tasks. In both these cases, one would expect schedules with smaller or the same makespan. But, that is not always the case. Chapter 2 presents problem instances where anomalies occur. The main reason for discussing anomalies now is that even today numerous papers are being published and systems are being deployed where "common sense"-based procedures are being introduced without any analytical justification or thorough experimental validation. Anomalies show that since we live for the most part in a "suboptimal world," the effect of our decisions is not always the intended one.

Other classical problems with numerous applications are the traveling salesperson, Steiner tree, and spanning tree problems, which will be defined later on. Even before the 1960s, there were several well-known polynomial time algorithms to construct minimum-weight spanning trees for edge-weighted graphs [2]. These simple greedy algorithms have low-order polynomial time complexity bounds. It was well known at that time that the same type of procedures do not always generate an optimal tour for the traveling salesperson problem (TSP), and do not always construct optimal Steiner trees. However, in 1968 E. F. Moore (see Ref. [3]) showed that for any set of points P in metric space $L_M < L_T \leq 2L_S$, where L_M, L_T, and L_S are the weights of a minimum-weight spanning tree, a minimum-weight tour (solution) for the TSP and minimum-weight Steiner tree for P, respectively. Since every spanning tree is a Steiner tree, the above bounds show that when using a minimum-weight spanning tree to approximate a minimum weight Steiner tree we have a solution (tree) whose weight is at most twice the weight of an optimal Steiner tree. In other words, any algorithm that generates a minimum-weight spanning tree is a 2-approximation algorithm for the Steiner tree problem. Furthermore, this approximation algorithm takes the same time as an algorithm that constructs a minimum-weight spanning tree for edge-weighted graphs [2], since such an algorithm can be used to construct an optimal spanning tree for a set of points in metric space. The above bound is established by defining a transformation from any minimum-weight Steiner tree into a TSP tour with weight at most $2L_S$. Therefore, $L_T \leq 2L_S$ [3]. Then by observing that the deletion of an edge in an optimum tour for the TSP results in a spanning tree, it follows that $L_M < L_T$. Chapter 3 discusses this approximation algorithm in detail. The Steiner ratio is defined as L_S/L_M. The above arguments show that the Steiner ratio is at least $\frac{1}{2}$. Gilbert and Pollak [3] conjectured that the Steiner ratio in the Euclidean plane equals $\frac{\sqrt{3}}{2}$ (the 0.86603 . . . conjecture). The proof of this conjecture and improved approximation algorithms for different versions of the Steiner tree problem are discussed in Chapters 42.

The above constructive proof can be applied to a minimum-weight spanning tree to generate a tour for the TSP. The construction takes polynomial time and results in a 2-approximation algorithm for the TSP. This approximation algorithm for the TSP is also referred to as the *double spanning tree algorithm* and is discussed in Chapters 3 and 31. Improved approximation algorithms for the TSP as well as algorithms for its generalizations are discussed in Chapters 3, 31, 40, 41, and 51. The approximation algorithm for the Steiner tree problem just discussed is explained in Chapter 3 and improved approximation algorithms and applications are discussed in Chapters 42, 43, and 51. Chapter 59 discusses approximation algorithms for variations of the spanning tree problem.

In 1969, Graham [4] studied the problem of scheduling tasks on identical machines, but restricted to independent tasks, i.e., the set of precedence constraints is empty. He analyzes the longest processing time (LPT) scheduling rule; this is list scheduling where the list of tasks L is arranged in nonincreasing order of their processing requirements. His elegant proof established that the LPT procedure generates a schedule with makespan at most $\frac{4}{3}-\frac{1}{3m}$ times the makespan of an optimal schedule, i.e., the LPT scheduling algorithm has a $\frac{4}{3}-\frac{1}{3m}$ approximation ratio. He also showed that the analysis is best possible for all m and $n \geq 2m + 1$. For $n \leq 2m$ tasks, the approximation ratio is smaller and under some conditions LPT generates an optimal makespan schedule. Graham [4], following a suggestion by D. Kleitman and D. Knuth, considered list schedules where the first portion of the list L consists of k tasks with the longest processing times arranged by their starting times in an optimal schedule for these k tasks (only). Then the list L has the remaining $n - k$ tasks in any order. The approximation ratio for this list schedule using list L is $1 + \frac{1-1/m}{1+\lceil k/m \rceil}$. An optimal schedule for the longest k tasks can be constructed in $O(km^k)$ time by a straightforward branch-and-bound algorithm. In other words, this algorithm has approximation ratio $1 + \epsilon$ and time complexity $O(n \log m + m^{(m-1-\epsilon m)/\epsilon})$. For any fixed constants m and ϵ, the algorithm constructs in polynomial (linear) time with respect to n a schedule with makespan at most $1 + \epsilon$ times the optimal makespan. Note that for a fixed constant m, the time complexity is polynomial with respect to n, but it is not polynomial with respect to $1/\epsilon$. This was the first algorithm of its kind and later on it was called a *polynomial time approximation scheme*. Chapter 9 discusses different PTASs. Additional PTASs appear in Chapters 42, 45, and 51. The proof techniques presented in Refs. [1,4] are outlined in Chapter 2, and have been extended to apply to other problems. There is an extensive body of literature for approximation algorithms and metaheuristics for scheduling problems. Chapters 44, 45, 46, 47, 73, and 81 discuss interesting approximation algorithms and heuristics for scheduling problems. The recent scheduling handbook [5] is an excellent source for scheduling algorithms, models, and performance analysis.

The development of NP-completeness theory in the early 1970s by Cook [6] and Karp [7] formally introduced the notion that there is a large class of decision problems (the answer to these problems is a simple yes or no) that are computationally equivalent. By this, it is meant that either every problem in this class has a polynomial time algorithm that solves it, or none of them do. Furthermore, this question is the same as the $P = NP$ question, an open problem in computational complexity. This question is to determine whether or not the set of languages recognized in polynomial time by deterministic Turing machines is the same as the set of languages recognized in polynomial time by nondeterministic Turing machines. The conjecture has been that $P \neq NP$, and thus the hardest problems in *NP* cannot be solved in polynomial time. These computationally equivalent problems are called *NP-complete* problems. The scheduling on identical machines problem discussed earlier is an optimization problem. Its corresponding decision problem has its input augmented by an integer value B and the yes-no question is to determine whether or not there is a schedule with makespan at most B. An optimization problem whose corresponding decision problem is NP-complete is called an *NP-hard* problem. Therefore, scheduling tasks on identical machines is an NP-hard problem. The TSP and the Steiner tree problem are also NP-hard problems. The minimum-weight spanning tree problem can be solved in polynomial time and is not an NP-hard problem under the assumption that $P \neq NP$. The next section discusses NP-completeness in more detail. There is a long list of practical problems arising in many different fields of study that are known to be NP-hard problems [8]. Because of this, the need to cope with these computationally intractable problems was recognized earlier on. This is when approximation algorithms became a central area of research activity. Approximation algorithms offered a way to circumvent computational intractability by paying a price when it comes to the quality of the solution generated. But a solution can be generated quickly. In other words and another language, “no te fijes en lo bien, fijate en lo rápido.” Words that my mother used to describe my ability to play golf when I was growing up.

In the early 1970s Garey et al. [9] as well as Johnson [10,11] developed the first set of polynomial time approximation algorithms for the bin packing problem. The analysis of the approximation ratio for these algorithms is asymptotic, which is different from those for the scheduling problems discussed earlier. We will define this notion precisely in the next section, but the idea is that the ratio holds when the optimal solution value is greater than some constant. Research on the bin packing problem and its variants has attracted very talented investigators who have generated more than 650 papers, most of which deal with approximations. This work has been driven by numerous applications in engineering and information sciences (see Chapters 32–35).

Johnson [12] developed polynomial time algorithms for the sum of subsets, max satisfiability, set cover, graph coloring, and max clique problems. The algorithms for the first two problems have a constant ratio approximation, but for the other problems the approximation ratio is ln n and n^{ϵ}. Sahni [13,14] developed a PTAS for the knapsack problem. Rosenkrantz et al. [15] developed several constant ratio approximation algorithms for the TSP. This version of the problem is defined over edge-weighted complete graphs that satisfy the triangle inequality (or simply metric graphs), rather than for points in metric space as in Ref. [3]. These algorithms have an approximation ratio of 2.

Sahni and Gonzalez [16] showed that there were a few NP-hard optimization problems for which the existence of a constant ratio polynomial time approximation algorithm implies the existence of a polynomial time algorithm to generate an optimal solution. In other words, for these problems the complexity of generating a constant ratio approximation and an optimal solution are computationally equivalent problems. For these problems, the approximation problem is NP-hard or simply inapproximable (under the assumption that $P \neq NP$). Later on, this notion was extended to mean that there is no polynomial time algorithm with approximation ratio r for a problem under some complexity theoretic hypothesis. The approximation ratio r is called the *in-approximability ratio*, and r may be a function of the input size (see Chapter 17).

The k-min-cluster problem is one of these inapproximable problems. Given an edge-weighted undirected graph, the k-min-cluster problem is to partition the set of vertices into k sets so as to minimize the sum of the weight of the edges with endpoints in the same set. The k-maxcut problem is defined as the k-min-cluster problem, except that the objective is to maximize the sum of the weight of the edges with endpoints in different sets. Even though these two problems have exactly the same set of feasible and optimal solutions, there is a linear time algorithm for the k-maxcut problem that generates k-cuts with weight at least $\frac{k-1}{k}$ times the weight of an optimal k-cut [16], whereas approximating the k-min-cluster problem is a computationally intractable problem. The former problem has the property that a near-optimal solution may be obtained as long as partial decisions are made optimally, whereas for the k-min-cluster an optimal partial decision may turn out to force a terrible overall solution.

Another interesting problem whose approximation problem is NP-hard is the TSP [16]. This is not exactly the same version of the TSP discussed above, which we said has several constant ratio polynomial time approximation algorithms. Given an edge-weighted undirected graph, the TSP is to find a least weight tour, i.e., find a least weight (simple) path that starts at vertex 1, visits each vertex in the graph *exactly* once, and ends at vertex 1. The weight of a path is the sum of the weight of its edges. The version of the TSP studied in Ref. [15] is limited to metric graphs, i.e., the graph is complete (all the edges are present) and the set of edge weights satisfies the triangle inequality (which means that the weight of the edge joining vertex i and j is less than or equal to the weight of any path from vertex i to vertex j). This version of the TSP is equivalent to the one studied by E. F. Moore [3]. The approximation algorithms given in Refs. [3,15] can be adapted easily to provide a constant-ratio approximation to the version of the TSP where the tour is defined as visiting each vertex in the graph *at least* once. Since Moore’s approximation algorithms for the metric Steiner tree and metric TSP are based on the same idea, one would expect that the Steiner tree problem defined over arbitrarily weighted graphs is NP-hard to approximate. However, this is not the case. Moore’s algorithm [3] can be modified to be a 2-approximation algorithm for this more general Steiner tree problem.

As pointed out in Ref. [17], Levner and Gens [18] added a couple of problems to the list of problems that are NP-hard to approximate. Garey and Johnson [19] showed that the max clique problem has the property that if for some constant r there is a polynomial time r-approximation algorithm, then there is a polynomial time r'-approximation algorithm for any constant r' such that $0 < r' < 1$. Since at that time researchers had considered many different polynomial time algorithms for the clique problem and none had a constant ratio approximation, it was conjectured that none existed, under the assumption that $P \neq NP$. This conjecture has been proved (see Chapter 17).

A PTAS is said to be an FPTAS if its time complexity is polynomial with respect to n (the problem size) and $1/\epsilon$. The first FPTAS was developed by Ibarra and Kim [20] for the knapsack problem. Sahni [21] developed three different techniques based on rounding, interval partitioning, and separation to construct FPTAS for sequencing and scheduling problems. These techniques have been extended to other problems and are discussed in Chapter 10. Horowitz and Sahni [22] developed FPTAS for scheduling on processors with different processing speed. Reference [17] discusses a simple $O(n^3/\epsilon)$ FPTAS for the knapsack problem developed by Babat [23,24]. Lawler [25] developed techniques to speed up FPTAS for the knapsack and related problems. Chapter 10 presents different methodologies to design FPTAS. Garey and Johnson [26] showed that if any problem in a class of NP-hard optimization problems that satisfy certain properties has a FPTAS, then $P = NP$. The properties are that the objective function value of every feasible solution is a positive integer, and the problem is *strongly* NP-hard. Strongly *NP*-hard means that the problem is NP-hard even when the magnitude of the maximum number in the input is bounded by a polynomial on the input length. For example, the TSP is strongly NP-hard, whereas the knapsack problem is not, under the assumption that $P \neq NP$ (see Chapter 10).

Lin and Kernighan [27] developed elaborate heuristics that established experimentally that instances of the TSP with up to 110 cities can be solved to optimality with 95% confidence in $O(n^2)$ time. This was an iterative improvement procedure applied to a set of randomly selected feasible solutions. The process was to perform k pairs of link (edge) interchanges that improved the length of the tour. However, Papadimitriou and Steiglitz [28] showed that for the TSP no local optimum of an efficiently searchable neighborhood can be within a constant factor of the optimum value unless $P = NP$. Since then, there has been quite a bit of research activity in this area. Deterministic and stochastic local search in efficiently searchable as well as in very large neighborhoods are discussed in Chapters 18–21. Chapter 14 discusses issues relating to the empirical evaluation of approximation algorithms and metaheuristics.

Perhaps the best known approximation algorithm is the one by Christofides [29] for the TSP defined over metric graphs. The approximation ratio for this algorithm is $\frac{3}{2}$, which is smaller than the approximation ratio of 2 for the algorithms reported in Refs. [3,15]. However, looking at the bigger picture that includes the time complexity of the approximation algorithms, Christofides algorithm is not of the same order as the ones given in Refs. [3,15]. Therefore, neither set of approximation algorithms dominates the other as one set has a smaller time complexity bound, whereas the other (Christofides algorithm) has a smaller worst-case approximation ratio.

Ausiello et al. [30] introduced the differential ratio, which is another way of measuring the quality of the solutions generated by approximation algorithms. The differential ratio destroys the artificial dissymmetry between "equivalent" minimization and maximization problems (e.g., the k-max cut and the k-min-cluster discussed above) when it comes to approximation. This ratio uses the difference between the worst possible solution and the solution generated by the algorithm, divided by the difference between the worst solution and the best solution. Cornuejols et al. [31] also discussed a variation of the differential ratio approximations. They wanted the ratio to satisfy the following property: "A modification of the data that adds a constant to the objective function value should also leave the error measure unchanged." That is, the "error" by the approximation algorithm should be the same as before. Differential ratio and its extensions are discussed in Chapter 16, along with other similar notions [30]. Ausiello et al. [30] also introduced *reductions that preserve approximability*. Since then, there have been several new types of approximation preserving reductions. The main advantage of these reductions is that they enable us to define large classes of optimization problems that behave in the same way with respect to approximation. Informally, the class of NP-optimization problems, **NPO**, is the set of all optimization problems Π that can be "recognized" in polynomial time (see Chapter 15 for a formal definition). An **NPO** problem Π is said to be in **APX**, if it has a constant approximation ratio polynomial time algorithm. The class **PTAS** consists of all **NPO** problems that have PTAS. The class **FPTAS** is defined similarly. Other classes, **Poly-APX**, **Log-APX**, and **Exp-APX**, have also been defined (see Chapter 15).

One of the main accomplishments at the end of the 1970s was the development of a polynomial time algorithm for linear programming problems by Khachiyan [32]. This result had a tremendous impact on approximation algorithms research, and started a new wave of approximation algorithms. Two subsequent research accomplishments were at least as significant as Khachiyan's [32] result. The first one was a faster polynomial time algorithm for solving linear programming problems developed by Karmakar [33]. The other major accomplishment was the work of Grötschel et al. [34,35]. They showed that it is possible to solve a linear programming problem with an exponential number of constraints (with respect to the number of variables) in time which is polynomial in the number of variables and the number of bits used to describe the input, given a *separation oracle* plus a bounding ball and a lower bound on the volume of the feasible solution space. Given a solution, the separation oracle determines in polynomial time whether or not the solution is feasible, and if it is not it finds a constraint that is violated. Chapter 11 gives an example of the use of this approach. Important developments have taken place during the past 20 years. The books [35,36] are excellent references for linear programming theory, algorithms, and applications.

Because of the above results, the approach of formulating the solution to an NP-hard problem as an integer linear programming problem and then solving the corresponding linear programming problem became very popular. This approach is discussed in Chapter 2. Once a fractional solution is obtained, one uses rounding to obtain a solution to the original NP-hard problem. The rounding may be deterministic or randomized, and it may be very complex (metarounding). LP rounding is discussed in Chapters 2, 4, 6–9, 11, 12, 37, 45, 57, 58, and 70.

Independently, Johnson [12] and Lovász [37] developed efficient algorithms for the set cover with approximation ratio of $1 + \ln d$, where d is the maximum number of elements in each set. Chvátal [38] extended this result to the weighted set cover problem. Subsequently, Hochbaum [39] developed an algorithm with approximation ratio f, where f is the maximum number of sets containing any of the elements in the set. This result is normally inferior to the one by Chvátal [38], but is more attractive for the weighted vertex cover problem, which is a restricted version of the weighted set cover. For this subproblem, it is a 2-approximation algorithm. A few months after Hochbaum's initial result,[1] Bar-Yehuda and Even [40] developed a primal-dual algorithm with the same approximation ratio as the one in [39]. The algorithm in [40] does not require the solution of an LP problem, as in the case of the algorithm in [39], and its time complexity is linear. But it uses linear programming theory. This was the first primal-dual approximation algorithm, though some previous algorithms may also be viewed as falling into this category. An application of the primal-dual approach, as well as related ones, is discussed in Chapter 2. Chapters 4, 37, 39, 40, and 71 discuss several primal-dual approximation algorithms. Chapter 13 discusses "distributed" primal-dual algorithms. These algorithms make decisions by using only "local" information.

In the mid 1980s, Bar-Yehuda and Even [41] developed a new framework parallel to the primal-dual methods. They call it *local ratio*; it is simple and requires no prior knowledge of linear programming. In Chapter 2, we explain the basics of this approach, and recent developments are discussed in [42].

Raghavan and Thompson [43] were the first to apply randomized rounding to relaxations of linear programming problems to generate solutions to the problem being approximated. This field has grown tremendously. LP randomized rounding is discussed in Chapters 2, 4, 6–8, 11, 12, 57, 70, and 80 and deterministic rounding is discussed in Chapters 2, 6, 7, 9, 11, 37, 45, 57, 58, and 70. A disadvantage of LP-rounding is that a linear programming problem needs to be solved. This takes polynomial time with

[1] Here, we are referring to the time when these results appeared as technical reports. Note that from the journal publication dates, the order is reversed. You will find similar patterns throughout the chapters. To add to the confusion, a large number of papers have also been published in conference proceedings. Since it would be very complex to include the dates when the initial technical report and conference proceedings were published, we only include the latest publication date. Please keep this in mind when you read the chapters and, in general, the computer science literature.

respect to the input length, but in this case it means the number of bits needed to represent the input. In contrast, algorithms based on the primal-dual approach are for the most part faster, since they take polynomial time with respect to the number of "objects" in the input. However, the LP-rounding approach can be applied to a much larger class of problems and it is more robust since the technique is more likely to be applicable after changing the objective function or constraints for a problem.

The first APTAS (asymptotic PTAS) was developed by Fernandez de la Vega and Lueker [44] for the bin packing problem. The first AFPTAS (Asymptotic FPTAS) for the same problem was developed by Karmakar and Karp [45]. These approaches are discussed in Chapter 16. Fully polynomial randomized approximation schemes (FPRAS) are discussed in Chapter 12.

In the 1980s, new approximation algorithms were developed as well as PTAS and FPTAS based on different approaches. These results are reported throughout the handbook. One difference was the application of approximation algorithms to other areas of research activity (very large-scale integration (VLSI), bioinformatics, network problems) as well as to other problems in established areas.

In the late 1980s, Papadimitriou and Yannakakis [46] defined *MAXSNP* as a subclass of NPO. These problems can be approximated within a constant factor and have a nice logical characterization. They showed that if MAX3SAT, vertex cover, MAXCUT, and some other problems in the class could be approximated in polynomial time with an arbitrary precision, then all MAXSNP problems have the same property. This fact was established by using *approximation preserving* reductions (see Chapters 15 and 17). In the 1990s, Arora et al. [47], using complex arguments (see Chapter 17), showed that MAX3SAT is hard to approximate within a factor of $1 + \epsilon$ for some $\epsilon > 0$ unless $P = NP$. Thus, all problems in MAXSNP do not admit a PTAS unless $P = NP$. This work led to major developments in the area of approximation algorithms, including inapproximability results for other problems, a bloom of approximation preserving reductions, discovery of new inapproximability classes, and construction of approximation algorithms achieving optimal or near optimal approximation ratios.

Feige et al. [48] showed that the clique problem could not be approximated to within some constant value. Applying the previous result in Ref. [26] showed that the clique problem is inapproximable to within any constant. Feige [49] showed that the set cover is inapproximable within ln n. Other inapproximable results appear in Refs. [50,51]. Chapter 17 discusses all of this work in detail.

There are many other very interesting results that have been published in the past 15 years. Goemans and Williamson [52] developed improved approximation algorithms for the maxcut and satisfiability problems using *semidefinite programming* (SDP). This seminal work opened a new venue for the design of approximation algorithms. Chapter 15 discusses this work as well as recent developments in this area. Goemans and Williamson [53] also developed powerful techniques for designing approximation algorithms based on the primal-dual approach. The dual-fitting and factor revealing approach is used in Ref. [54]. Techniques and extensions of these approaches are discussed in Chapters 4, 13, 37, 39, 40, and 71.

In the past couple of decades, we have seen approximation algorithms being applied to traditional combinatorial optimization problems as well as problems arising in other areas of research activity. These areas include VLSI design automation, networks (wired, sensor and wireless), bioinformatics, game theory, computational geometry, and graph problems. In Section 2, we elaborate further on these applications.

1.2.2 Local Search, Artificial Neural Networks, and Metaheuristics

Local search techniques have a long history; they range from simple constructive and iterative improvement algorithms to rather complex methods that require significant fine-tuning, such as evolutionary algorithms (EAs) or SA. Local search is perhaps one of the most natural ways to attempt to find an optimal or suboptimal solution to an optimization problem. The idea of local search is simple: start from a solution and improve it by making local changes until no further progress is possible. Deterministic local search algorithms are discussed in Chapter 18. Chapter 19 covers stochastic local search algorithms. These are local search algorithms that make use of randomized decisions, for example, in the context of generating initial solutions or when determining search steps. When the neighborhood to search for the next solution is very large, finding the best neighbor to move to is many times an NP-hard problem. Therefore, a suboptimal solution is needed at this step. In Chapter 20, the issues related to very large-scale neighborhood search are discussed from the theoretical, algorithmic, and applications point of view.

Reactive search advocates the use of simple sub symbolic machine learning to automate the parameter tuning process and make it an integral (and fully documented) part of the algorithm. Parameters are normally tuned through a feedback loop that many times depends on the user. Reactive search attempts to mechanize this process. Chapter 21 discusses issues arising during this process.

Artificial neural networks have been proposed as a tool for machine learning and many results have been obtained regarding their application to practical problems in robotics control, vision, pattern recognition, grammatical inferences, and other areas. Once trained, the network will compute an input/output mapping that, if the training data was representative enough, will closely match the unknown rule that produced the original data. Neural networks are discussed in Chapter 22.

The work of Lin and Kernighan [27] as well as that of others sparked the study of modern heuristics, which have evolved and are now called *metaheuristics*. The term metaheuristics was coined by Glover [55] in 1986 and in general means “to find beyond in an upper level.” Metaheuristics include Tabu Search (TS), Simulated Annealing (SA), Ant Colony Optimization, Evolutionary Computation (EC), iterated local search (ILC), and Memetic Algorithms (MA). One of the motivations for the study of metaheuristics is that it was recognized early on that constant ratio polynomial time approximation algorithms are not likely to exist for a large class of practical problems [16]. Metaheuristics do not guarantee that near-optimal solutions will be found quickly for all problem instances. However, these complex programs do find near-optimal solutions for many problem instances that arise in practice. These procedures have a wide range of applicability. This is the most appealing aspect of metaheuristics.

The term Tabu Search (TS) was coined by Glover [55]. TS is based on *adaptive memory* and *responsive exploration*. The former allows for the effective and efficient search of the solution space. The latter is used to guide the search process by imposing restraints and inducements based on the information collected. Intensification and diversification are controlled by the information collected, rather than by a random process. Chapter 23 discusses many different aspects of TS as well as problems to which it has been applied.

In the early 1980s Kirkpatrick et al. [56] and, independently, Černý [57] introduced Simulated Annealing (SA) as a randomized local search algorithm to solve combinatorial optimization problems. *SA* is a local search algorithm, which means that it starts with an initial solution and then searches through the solution space by iteratively generating a new solution that is “near” it. Sometimes, the moves are to a worse solution to escape local optimal solutions. This method is based on statistical mechanics (Metropolis algorithm). It was heavily inspired by an analogy between the physical annealing process of solids and the problem of solving large combinatorial optimization problems. Chapter 25 discusses this approach in detail.

Evolutionary Computation (EC) is a metaphor for building, applying, and studying algorithms based on Darwinian principles of natural selection. Algorithms that are based on evolutionary principles are called *evolutionary algorithms* (EA). They are inspired by nature’s capability to evolve living beings well adapted to their environment. There has been a variety of slightly different EAs proposed over the years. Three different strands of EAs were developed independently of each other over time. These are *evolutionary programming* (EP) introduced by Fogel [58] and Fogel et al. [59], *evolutionary strategies* (ES) proposed by Rechenberg [60], and *genetic algorithms* (GAs) initiated by Holland [61]. GAs are mainly applied to solve discrete problems. *Genetic programming* (GP) and *scatter search* (SS) are more recent members of the EA family. EAs can be understood from a unified point of view with respect to their main components and the way they explore the search space. EC is discussed in Chapter 24.

Chapter 17 presents an overview of Ant Colony Optimization (ACO)—a metaheuristic inspired by the behavior of real ants. ACO was proposed by Dorigo and colleagues [62] in the early 1990s as a method for solving hard combinatorial optimization problems. ACO algorithms may be considered to be part of *swarm intelligence*, the research field that studies algorithms inspired by the observation of the behavior of *swarms*. Swarm intelligence algorithms are made up of simple individuals that cooperate through self-organization.

Memetic Algorithms (MA) were introduced by Moscato [63] in the late 1980s to denote a family of metaheuristics that can be characterized as the hybridization of different algorithmic approaches for a given problem. It is a population-based approach in which a set of cooperating and competing agents are engaged in periods of individual improvement of the solutions while they sporadically interact. An important component is *problem and instance-dependent knowledge*, which is used to speed-up the search process. A complete description is given in Chapter 27.

1.2.3 Sensitivity Analysis, Multiobjective Optimization, and Stability

Chapter 30 covers *sensitivity analysis*, which has been around for more than 40 years. The aim is to study how variations affect the optimal solution value. In particular, parametric analysis studies problems whose structure is fixed, but where cost coefficients vary continuously as a function of one or more parameters. This is important when selecting the model parameters in optimization problems. In contrast, Chapter 31 considers a newer area, which is called *stability*. By this we mean how the complexity of a problem depends on a parameter whose variation alters the space of allowable instances.

Chapters 28 and 29 discuss *multiobjective combinatorial optimization*. This is important in practice since quite often a decision is rarely made with only one criterion. There are many examples of such applications in the areas of transportation, communication, biology, finance, and also computer science. Approximation algorithms and a FPTAS for multiobjective optimization problems are discussed in Chapter 28. Chapter 29 covers stochastic local search algorithms for multiobjective optimization problems.

1.3 Definitions and Notation

One can use many different criteria to judge approximation algorithms and heuristics. For example the quality of solution generated, and the time and space complexity needed to generate it. One may measure the criteria in different ways, e.g., we could use the worst case, average case, median case, etc. The evaluation could be analytical or experimental. Additional criteria include characterization of data sets where the algorithm performs very well or very poorly; comparison with other algorithms using benchmarks or data sets arising in practice; tightness of bounds (for quality of solution, time and space complexity); the value of the constants associated with the time complexity bound including the ones for the lower order terms; and so on. For some researchers, the most important aspect of an approximation algorithm is that it is complex to analyze, but for others it is more important that the algorithm be complex and involve the use of sophisticated data structures. For researchers working on problems directly applicable to the "real world," experimental evaluation or evaluation on benchmarks is a more important criterion. Clearly, there is a wide variety of criteria one can use to evaluate approximation algorithms. The chapters in this handbook use different criteria to evaluate approximation algorithms.

For any given optimization problem P, let $A_1, A_2, \ldots$ be the set of current algorithms that generate a feasible solution for each instance of problem P. Suppose that we select a set of criteria C and a way to measure it that we feel is the most important. How can we decide which algorithm is best for problem P with respect to C? We may visualize every algorithm as a point in multidimensional space. Now, the approach used to compare feasible solutions for multiobjective function problems (see Chapters 28 and 29) can also be used in this case to label some of the algorithms as current Pareto optimal with respect to C. Algorithm A is said to be *dominated* by algorithm B with respect to C, if for each criterion $c \in C$ algorithm B is "not worse" than A, and for at least one criterion $c \in C$ algorithm B is "better" than A. An algorithm is said to be a *current Pareto optimal* algorithm with respect to C if none of the current algorithms dominates it.

In the next subsections, we define time and space complexity, NP-completeness, and different ways to measure the quality of the solutions generated by the algorithms.

1.3.1 Time and Space Complexity

There are many different ways one can use to judge algorithms. The main ones we use are the time and space required to solve the problem. This can be expressed in terms on n, the input size. It can be evaluated empirically or analytically. For the analytical evaluation, we use the time and space complexity of the algorithm. Informally, this is a way to express the time the algorithm takes to solve a problem of size n and the amount of space needed to run the algorithm.

It is clear that almost all algorithms take different time to execute with different data sets even when the input size is the same. If you code it and run it on a computer you will see more variation depending on the different hardware and software installed in the system. It is impossible to characterize exactly the time and space required by an algorithm. We need a short cut. The approach that has been taken is to count the number of "operations" performed by the algorithm in terms of the input size. "Operations" is not an exact term and refers to a set of "instructions" whose number is independent of the problem size. Then we just need to count the total number of operations.

Counting the number of operations exactly is very complex for a large number of algorithms. So we just take into consideration the "highest"-order term. This is the O notation.

Big "oh" notation: A (positive) function $f(n)$ is said to be $O(g(n))$ if there exist two constants $c \geq 1$ and $n_0 \geq 1$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

The function $g(n)$ is the highest-order term. For example, if $f(n) = n^3 + 20n^2$, then $g(n) = n^3$. Setting $n_0 = 1$ and $c = 21$ shows that $f(n)$ is $O(n^3)$. Note that $f(n)$ is also $O(n^4)$, but we like $g(n)$ to be the function with the smallest possible growth. The function $f(n)$ cannot be $O(n^2)$ because it is impossible to find constants c and n_0 such that $n^3 + 20n^2 \leq cn^2$ for all $n \geq n_0$.

The time and space complexity of an algorithm is expressed in the O notation and describes their growth rate in terms of the problem size. Normally, the problem size is the number of vertices and edges in a graph, the number of tasks and machines in a scheduling problem, etc. But it can also be the number of bits used to represent the input.

When comparing two algorithms expressed in O notation, we have to be careful because the constants c and n_0 are hidden. For large n, the algorithm with the smallest growth rate is the better one. When two algorithms have similar constants c and n_0, the algorithm with the smallest growth function has a smaller running time. The book [2] discusses in detail the O notation as well as other notation.

1.3.2 NP-Completeness

Before the 1970s, researchers were aware that some problems could be computationally solved by algorithms with (low) polynomial time complexity ($O(n)$, $O(n^2)$, $O(n^3)$, etc.), whereas other problems had exponential time complexity, for example, $O(2^n)$ and $O(n!)$. It was clear that even for small values of n, exponential time complexity equates to computational intractability if the algorithm actually performs an exponential number of operations for some inputs. The convention of computational tractability being equated to polynomial time complexity does not really fit well, as an algorithm with time complexity $O(n^{100})$ is not really tractable if it actually performs n^{100} operations. But even under this relaxation of "tractability," there is a large class of problems that does not seem to have computationally tractable algorithms.

We have been discussing optimization problems. But NP-completeness is defined with respect to decision problems. A decision problem is simply one whose answer is "yes" or "no." The scheduling on identical machines problems discussed earlier is an optimization problem. Its corresponding decision problem has its input augmented by an integer value B and the yes-no question is to determine whether or not there is a schedule with makespan at most B. Every optimization problem has a corresponding decision problem. Since the solution of an optimization problem can be used directly to solve the decision problem, we say that the optimization problem is at least as hard to solve as the decision problem. If we show that the decision problem is a computationally intractable problem, then the corresponding optimization problem is also intractable.

The development of NP-completeness theory in the early 1970s by Cook [6] and Karp [7] formally introduced the notion that there is a large class of decision problems that are computationally equivalent. By this we mean that either every problem in this class has a polynomial time algorithm that solves it, or none of them do. Furthermore, this question is the same as the $P = NP$ question, an open problem in computational complexity. This question is to determine whether or not the set of languages recognized in polynomial time by deterministic Turing machines is the same as the set of languages recognized in polynomial time by nondeterministic Turing machines. The conjecture has been that $P \neq NP$, and thus the problems in this class do not have polynomial time algorithms for their solution. The decision problems in this class of problems are called *NP-complete* problems. Optimization problems whose corresponding decision problems are NP-complete are called *NP-hard* problems.

Scheduling tasks on identical machines is an NP-hard problem. The TSP and Steiner tree problem are also NP-hard problems. The minimum-weight spanning tree problem can be solved in polynomial and it is not an NP-hard problem, under the assumption that $P \neq NP$. There is a long list of practical problems arising in many different fields of study that are known to be NP-hard problems. In fact, almost all the optimization problems discussed in this handbook are NP-hard problems. The book [8] is an excellent source of information for NP-complete and NP-hard problems.

One establishes that a problem Q is an NP-complete problem by showing that the problem is in NP and giving a polynomial time transformation from an NP-complete problem to the problem Q.

A problem is said to be in *NP* if one can show that a yes answer to it can be verified in polynomial time. For the scheduling problem defined above, you may think of this as providing a procedure that given any instance of the problem and an assignment of tasks to machines, the algorithm verifies in polynomial time, with respect to the problem instance size, that the assignment is a schedule and its makespan is at most B. This is equivalent to the task a grader does when grading a question of the form "Does the following instance of the scheduling problem have a schedule with makespan at most 300? If so, give a schedule." Just verifying that the "answer" is correct is a simple problem. But solving a problem instance with 10,000 tasks and 20 machines seems much harder than simply grading it. In our oversimplification, it seems that $P \neq NP$. Polynomial time verification of a yes answer does not seem to imply polynomial time solvability.

A polynomial time transformation from decision problem P_1 to decision problem P_2 is an algorithm that takes as input any instance I of problem P_1 and constructs an instance $f(I)$ of P_2. The algorithm must take polynomial time with respect to the size of the instance I. The transformation must be such that $f(I)$ is a yes-instance of P_2 if, and only if, I is a yes-instance of P_1.

The implication of a polynomial transformation $P_1 \alpha P_2$ is that if P_2 can be solved in polynomial time, then so can P_1, and if P_1 cannot be solved in polynomial time, then P_2 cannot be solved in polynomial time.

Consider the partition problem. We are given n items $1, 2, \ldots, n$. Item j has size $s(j)$. The problem is to determine whether or not the set of items can be partitioned into two sets such that the sum of the size of the items in one set equals the sum of the size of the items in the other set. Now let us polynomially transform the partition problem to the decision version of the identical machines scheduling problem. Given any instance I of partition, we define the instance $f(I)$ as follows. There are n tasks and $m = 2$ machines. Task i represents item i and its processing time is $s(i)$. All the tasks are independent and $B = \sum_{i=1}^{i=n} s(i)/2$. Clearly, $f(I)$ has a schedule with maskespan B iff the instance I has a partition.

A decision problem is said to be *strongly NP-complete* if the problem is NP-complete even when all the "numbers" in the problem instance are less than or equal to $p(n)$, where p is a polynomial and n is the "size" of the problem instance. Partition is not NP-complete in the strong sense (under the assumption that $P \neq NP$) because there is a polynomial time dynamic programming algorithm to solve this problem when $\sum s(i) \leq p(n)$ (see Chapter 10). An excellent source for NP-completeness information is the book by Garey and Johnson [8].

1.3.3 Performance Evaluation of Algorithms

The main criterion used to compare approximation algorithms has been the quality of the solution generated. Let us consider different ways to compare the quality of the solutions generated when measuring the worst case. That is the main criterion discussed in Section 1.2.

For some problems, it is very hard to judge the quality of the solution generated. For example, approximating colors, can only be judged by viewing the resulting images and that is subjective (see Chapter 86). Chapter 85 covers digital reputation schemes. Here again, it is difficult to judge the quality of the solution generated. Problems in the application areas of bioinformatics and VLSI fall into this category because, in general, these are problems with multiobjective functions.

In what follows, we concentrate on problems where it is possible to judge the quality of the solution generated. At this point, we need to introduce additional notation. Let P be an optimization problem and let A be an algorithm that generates a feasible solution for every instance I of problem P. We use $\hat{f}_A(I)$ to denote the objective function value of the solution generated by algorithm A for instance I. We drop A and use $\hat{f}(I)$ when it is clear which algorithm is being used. Let $f^*(I)$ be the objective function value of an optimal solution for instance I. Note that normally we do not know the value of $f^*(I)$ exactly, but we have bounds that should be as tight as possible.

Let G be an undirected graph that represents a set of cities (vertices) and roads (edges) between a pair of cities. Every edge has a positive number called the weight (or cost) and represents the cost of driving (gas plus tolls) between the pair of cities it joins. A *shortest path* from vertex s to vertex t in G is an st-path (path from s to t) such that the sum of the weight of the edges in it is the "'least possible among all possible st-paths." There are well-known algorithms that solve this shortest-path problem in polynomial time [2]. Let A be an algorithm that generates a feasible solution (st-path) for every instance I of problem P. If for every instance I, algorithm A generates an st-path such that

$$\hat{f}(I) \leq f^*(I) + c$$

where c is some fixed constant, then A is said to be an *absolute* approximation algorithm for problem P with (additive) approximation bound c. Ideally, we would like to design a linear (or at least polynomial) time approximation algorithm with the smallest possible approximation bound. It is not difficult to see that this is not a good way of measuring the quality of a solution. Suppose that we have a graph G and we are running an absolute approximation algorithm for the shortest path problem concurrently in two different countries with the edge weight expressed in the local currency. Furthermore, assume that there is a large exchange rate between the two currencies. Any approximation algorithm solving the weak currency instance will have a much harder time finding a solution within the bound of c, than when solving the strong currency instance. We can take this to the extreme. We now claim that the above absolute approximation algorithm A can be used to generate an optimal solution for every problem instance within the same time complexity bound.

The argument is simple. Given any instance I of the shortest-path problem, we construct an instance I_{c+1} using the same graph, but every edge weight is multiplied by $c + 1$. Clearly, $f^*(I_{c+1}) = (c + 1) f^*(I)$. The st-path for I_{c+1} constructed by the algorithm is also an st-path in I with weight $\hat{f}(I) = \hat{f}(I_{c+1})/(c + 1)$. Since $\hat{f}(I_{c+1}) \leq f^*(I_{c+1}) + c$, then by substituting the above bounds we know that

$$\hat{f}(I) = \frac{\hat{f}(I_{c+1})}{(c+1)} \leq \frac{f^*(I_{c+1})}{c+1} + \frac{c}{c+1} = f^*(I) + \frac{c}{c+1}$$

Since all the edges have integer weights, it then follows that the algorithm solves the problem optimally. In other words, for the shortest path problem any algorithm that generates a solution with (additive) approximation bound c can be used to generate an optimal solution within the same time complexity bound. This same property can be established for almost all NP-hard optimization problems. Because of this, the use of absolute approximation has never been given a serious consideration.

Sahni [14] defines as an ϵ-approximation algorithm for problem P an algorithm that generates a feasible solution for every problem instance I of P such that

$$|\frac{\hat{f}(I) - f^*(I)}{f^*(I)}| \leq \epsilon$$

It is assumed that $f^*(I) > 0$. For a minimization problem, $\epsilon > 0$ and for a maximization problem, $0 < \epsilon < 1$. In both cases, ϵ represents the percentage of error. The algorithm is called an ϵ-approximation algorithm and the solution is said to be an ϵ-approximate solution. Graham's list scheduling algorithm [1] is a $1-1/n$-approximation algorithm, and Sahni and Gonzalez [16] algorithm for the k-maxcut problem is a $\frac{1}{k}$-approximation algorithm (see Section 1.2). Note that this notation is different from the one discussed in Section 1.2. The difference is 1 unit, i.e., the ϵ in this notation corresponds to $1+\epsilon$ in the other.

Johnson [12] used a slightly different, but equivalent notation. He uses the approximation ratio ρ to mean that for every problem instance I of P, the algorithm satisfies $\frac{\hat{f}(I)}{f^*(I)} \leq \rho$ for minimization problems, and $\frac{f^*(I)}{\hat{f}(I)} \leq \rho$ for maximization problems. The one for minimization problems is the same as the one given in Ref. [1]. The value for ρ is always greater than 1, and the closer to 1, the better the solution generated by the algorithm. One refers to ρ as the *approximation ratio* and the algorithm is a ρ-approximation algorithm. The list scheduling algorithm in the previous section is a $(2-\frac{1}{m})$-approximation algorithm and the algorithm for the k-maxcut problem is a $(\frac{k}{k-1})$-approximation algorithm. Sometimes, $1/\rho$ is used as the approximation ratio for maximization problems. Using this notation, the algorithm for the k-maxcut problem in the previous section is a $1-\frac{1}{k}$-approximation algorithm.

All the above forms are in use today. The most popular ones are ρ for minimization and $1/\rho$ for maximization. These are referred to as approximation ratios or approximation factors. We refer to all these algorithms as ϵ-approximation algorithms. The point to remember is that one needs to be aware of the differences and be alert when reading the literature. In the above discussion, we make ϵ and ρ look as if they are fixed constants. But, they can be made dependent on the size of the problem instance I. For example, it may be $\ln n$, or n^{ϵ} for some problems, where n is some parameter of the problem that depends on I, e.g., the number of nodes in the input graph, and ϵ depends on the algorithm being used to generate the solutions.

Normally, one prefers an algorithm with a smaller approximation ratio. However, it is not always the case that an algorithm with smaller approximation ratio always generates solutions closer to optimal than one with a larger approximation ratio. The main reason is that the notation is for the worst-case ratio and the worst case does not always occur. But there are other reasons too. For example, the bound for the optimal solution value used in the analysis of two different algorithms may be different. Let P be the shortest-path minimization problem and let A be an algorithm with approximation ratio 2. In this case, we use d as the lower bound for $f^*(I)$, where d is some parameter of the problem instance. Algorithm B is a 1.5-approximation algorithm, but $f^*(I)$ used to establish it is the exact optimal solution value. Suppose that for problem instance I the value of d is 5 and $f^*(I) = 8$. Algorithm A will generate a path with weight at most 10, whereas algorithm B will generate one with weight at most $1.5 \times 8 = 12$. So the solution generated by Algorithm B may be worse than the one generated by A even if both algorithms generate the worst values for the instance. One can argue that the average "error" makes more sense than worst case. The problem is how to define and establish bounds for average "error." There are many other pitfalls when using worst-case ratios. It is important to keep all this in mind when making comparisons between algorithms. In practice, one may run several different approximation algorithms concurrently and output the best of the solutions. This has the disadvantage that the running time of this compound algorithm will be the one for the slowest algorithm.

There are a few problems for which the worst-case approximation ratio applies only to problem instances where the value of the optimal solution is small. One such problem is the bin packing problem discussed in Section 1.2. Informally, ρ_A^{∞} is the smallest constant such that there exists a constant $K < \infty$ for which

$$\hat{f}(I) \leq \rho_A^{\infty} f^*(I) + K$$

The *asymptotic approximation ratio* is the multiplicative constant and it hides the additive constant K. This is most useful when K is small. Chapter 32 discusses this notation formally. The asymptotic notation is mainly used for bin packing and some of its variants.

Ausiello et al. [30] introduced the *differential ratio*. Informally, an algorithm is said to be a δ differential ratio approximation algorithm if for every instance I of P

$$\frac{\omega(I) - \hat{f}(I)}{\omega(I) - f^*(I)} \leq \delta$$

where $\omega(I)$ is the value of a worst solution for instance I. Differential ratio has some interesting properties for the complexity of the approximation problems. Chapter 16 discusses differential ratio approximation and its variations.

As said earlier, there are many different criteria to compare algorithms. What if we use both the approximation ratio and time complexity? For example, the approximation algorithms in Ref. [15] and the one in Ref. [29] are current Pareto optimal with respect to these criteria for the TSP defined over metric graphs. Neither of the algorithms dominates the others in both time complexity and approximation ratio. The same can be said about the simple linear time approximation algorithm for the k-maxcut problem in Ref. [16] and the complex one given in Ref. [52] or the more recent ones that apply for all k.

The best algorithm to use also depends on the instance being solved. It makes a difference whether we are dealing with an instance of the TSP with optimal tour cost equal to a billion dollars and one with optimal cost equal to just a few pennies. Though, it also depends on the number of such instances being solved.

More elaborate approximation algorithms have been developed that generate a solution for any fixed constant ϵ. Formally, a PTAS for problem P is an algorithm A that given any fixed constant $\epsilon > 0$, it constructs a solution to problem P such that $|\frac{\hat{f}(I) - f^*(I)}{f^*(I)}| \leq \epsilon$ in polynomial time with respect to the length of the instance I. Note that the time complexity may be exponential with respect to $1/\epsilon$. For example, the time complexity could be $O(n^{(1/\epsilon)})$ or $O(n + 4^{O(1/\epsilon)})$. Equivalent PTAS are also defined using different notation, for example, based on $\frac{\hat{f}(I)}{f^*(I)} \leq 1 + \epsilon$ for minimization problems.

One would like to design PTAS for all problems, but that is not possible unless $P = NP$. Clearly, with respect to approximation ratios, the PTAS is better than the ϵ-approximation algorithms for some ϵ. But their main drawback is that they are not practical because the time complexity is exponential on $1/\epsilon$. This does not preclude the existence of a practical PTAS for "natural" occurring problems. However, a PTAS establishes that a problem can be approximated for all fixed constants. Different types of PTAS are discussed in Chapter 9. Additional PTAS are presented in Chapters 42, 45, and 51.

A PTAS is said to be an FPTAS if its time complexity is polynomial with respect to n (the problem size) and $1/\epsilon$. FPTAS are for the most part practical algorithms. Different methodologies for designing FPTAS are discussed in Chapter 10.

Approximation schemes based on asymptotic approximation and on randomized algorithms have been developed. Chapters 11 and 45 discuss asymptotic approximation schemes and Chapter 12 discusses randomized approximation schemes.

References

[1] Graham, R. L., Bounds for certain multiprocessing anomalies, *Bell System Tech. J.*, 45, 1563, 1966.

[2] Sahni, S., *Data Structures, Algorithms, and Applications in C++*, 2nd ed., Silicon Press, Summit, NJ, 2005.

[3] Gilbert, E. N. and Pollak, H. O., Steiner minimal trees, *SIAM J. Appl. Math.*, 16(1), 1, 1968.

[4] Graham, R. L., Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.*, 17, 263, 1969.

[5] Leung, J. Y.-T., Ed., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman & Hall/CRC, Boca Raton, FL, 2004.

[6] Cook, S. A., The complexity of theorem-proving procedures, *Proc. STOC'71*, 1971, p. 151.

[7] Karp, R. M., Reducibility among combinatorial problems, in R. E. Miller and J. W. Thatcher, eds., *Complexity of Computer Computations*, Plenum Press, New York, 1972, p. 85.

[8] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, NY, 1979.

[9] Garey, M. R., Graham, R. L., and Ullman, J. D., Worst-case analysis of memory allocation algorithms, *Proc. STOC*, ACM, 1972, p. 143.

[10] Johnson, D. S., Near-Optimal Bin Packing Algorithms, Ph.D. thesis, Massachusetts Institute of Technology, Department of Mathematics, Cambridge, 1973.

[11] Johnson, D. S., Fast algorithms for bin packing, *JCSS*, 8, 272, 1974.

[12] Johnson, D. S., Approximation algorithms for combinatorial problems, *JCSS*, 9, 256, 1974.

[13] Sahni, S., On the Knapsack and Other Computationally Related Problems, Ph.D. thesis, Cornell University, 1973.

[14] Sahni, S., Approximate algorithms for the 0/1 knapsack problem, *JACM*, 22(1), 115, 1975.

[15] Rosenkrantz, R., Stearns, R., and Lewis, L., An analysis of several heuristics for the traveling salesman problem, *SIAM J. Comput.*, 6(3), 563, 1977.

[16] Sahni, S. and Gonzalez, T., P-complete approximation problems, *JACM*, 23, 555, 1976.

[17] Gens, G. V. and Levner, E., Complexity of approximation algorithms for combinatorial problems: A survey, *SIGACT News*, 12(3), 52, 1980.

[18] Levner, E. and Gens, G. V., *Discrete Optimization Problems and Efficient Approximation Algorithms,* Central Economic and Mathematics Institute, Moscow, 1978 (in Russian).

[19] Garey, M. R. and Johnson, D. S., The complexity of near-optimal graph coloring, *SIAM J. Comput.*, 4, 397, 1975.

[20] Ibarra, O. and Kim, C., Fast approximation algorithms for the knapsack and sum of subset problems, *JACM*, 22(4), 463, 1975.

[21] Sahni, S., Algorithms for scheduling independent tasks, *JACM*, 23(1), 116, 1976.

[22] Horowitz, E. and Sahni, S., Exact and approximate algorithms for scheduling nonidentical processors, *JACM*, 23(2), 317, 1976.

[23] Babat, L. G., Approximate computation of linear functions on vertices of the unit N-dimensional cube, in *Studies in Discrete Optimization,* Fridman, A. A., Ed., Nauka, Moscow, 1976 (in Russian).

[24] Babat, L. G., A fixed-charge problem, *Izv. Akad. Nauk SSR, Techn, Kibernet.*, 3, 25, 1978 (in Russian).

[25] Lawler, E., Fast approximation algorithms for knapsack problems, *Math. Oper. Res.*, 4, 339, 1979.

[26] Garey, M. R. and Johnson, D. S., Strong NP-completeness results: Motivations, examples, and implications, *JACM*, 25, 499, 1978.

[27] Lin, S. and Kernighan, B. W., An effective heuristic algorithm for the traveling salesman problem, *Oper. Res.*, 21(2), 498, 1973.

[28] Papadimitriou, C. H. and Steiglitz, K., On the complexity of local search for the traveling salesman problem, *SIAM J. Comput.*, 6, 76, 1977.

[29] Christofides, N., Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem. Technical Report 338, Grad School of Industrial Administration, CMU, 1976.

[30] Ausiello, G., D'Atri, A., and Protasi, M., On the structure of combinatorial problems and structure preserving reductions, in *Proc. ICALP'77*, Lecture Notes in Computer Science, Vol. 52 Springer, Berlin, 1977, p. 45.

[31] Cornuejols, G., Fisher, M. L., and Nemhauser, G. L., Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms, *Manage. Sci.*, 23(8), 789, 1977.

[32] Khachiyan, L. G., A polynomial algorithms for the linear programming problem, *Dokl. Akad. Nauk SSSR*, 244(5), 1979 (in Russian).

[33] Karmakar, N., A new polynomial-time algorithm for linear programming, *Combinatorica,* 4, 373, 1984.

[34] Grötschel, M., Lovász, L., and Schrijver, A., The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica,* 1, 169, 1981.

[35] Schrijver, A., *Theory of Linear and Integer Programming,* Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley, New York, 2000.

[36] Vanderbei, R. J., *Linear Programming Foundations and Extensions,* Series: International Series in Operations Research & Management Science, Vol. 37, Springer, Berlin.

[37] Lovász, L., On the ratio of optimal integral and fractional covers, *Discrete Math.*, 13, 383, 1975.

[38] Chvátal, V., A greedy heuristic for the set-covering problem, *Math. Oper. Res.,* 4(3), 233, 1979.

[39] Hochbaum, D. S., Approximation algorithms for set covering and vertex covering problems, *SIAM J. Comput.*, 11, 555, 1982.

[40] Bar-Yehuda, R. and Even, S., A linear time approximation algorithm for the weighted vertex cover problem, *J. Algorithms*, 2, 198, 1981.

[41] Bar-Yehuda, R. and Even, S., A local-ratio theorem for approximating the weighted set cover problem, *Ann. of Disc. Math.*, 25, 27, 1985.

[42] Bar-Yehuda, R. and Bendel, K., Local ratio: A unified framework for approximation algorithms, *ACM Comput. Surv.*, 36(4), 422, 2004.

[43] Raghavan, R. and Thompson, C., Randomized rounding: A technique for provably good algorithms and algorithmic proof, *Combinatorica*, 7, 365, 1987.

[44] Fernandez de la Vega, W. and Lueker, G. S., Bin packing can be solved within $1 + \epsilon$ in linear time, *Combinatorica,* 1, 349, 1981.

[45] Karmakar, N. and Karp, R. M., An efficient approximation scheme for the one-dimensional bin packing problem, *Proc. FOCS,* 1982, p. 312.

[46] Papadimitriou, C. H. and Yannakakis, M., Optimization, approximation and complexity classes, *J. Comput. Syst. Sci.*, 43, 425, 1991.

[47] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M., Proof verification and hardness of approximation problems, *Proc. FOCS*, 1992.

[48] Feige, U., Goldwasser, S., Lovasz, L., Safra, S., and Szegedy, M., Interactive proofs and the hardness of approximating cliques, *JACM*, 43, 1996.

[49] Feige, U., A threshold of ln *n* for approximating set cover, *JACM*, 45(4), 634, 1998. (Prelim. version in STOC'96.)

[50] Engebretsen, L. and Holmerin, J., Towards optimal lower bounds for clique and chromatic number, *TCS*, 299, 2003.

[51] Hastad, J., Some optimal inapproximability results, *JACM*, 48, 2001. (Prelim. version in STOC'97.)

[52] Goemans, M. X. and Williamson, D. P., Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *JACM*, 42(6), 1115, 1995.

[53] Goemans, M. X. and Williamson, D. P., A general approximation technique for constrained forest problems, *SIAM J. Comput.*, 24(2), 296, 1995.

[54] Jain, K., Mahdian, M., Markakis, E., Saberi, A., and Vazirani, V. V., Approximation algorithms for facility location via dual fitting with factor-revealing LP, *JACM*, 50, 795, 2003.

[55] Glover, F., Future paths for integer programming and links to artificial intelligence, *Comput. Oper. Res.*, 13, 533, 1986.

[56] Kirkpatrick, S., Gelatt, C. D., Jr. and Vecchi, M. P., Optimization by simulated annealing, *Science,* 220, 671, 1983.

[57] Černý, V., Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm, *J. Optimization Theory Appl.*, 45, 41, 1985.

[58] Fogel, L. J., Toward inductive inference automata, in *Proc. Int. Fed. Inf. Process. Congr.*, 1962, 395.

[59] Fogel, L. J., Owens, A. J., and Walsh, M. J., *Artificial Intelligence through Simulated Evolution*, Wiley, New York, 1966.

[60] Rechenberg, I., *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog, Stuttgart, 1973.

[61] Holland, J. H., *Adaption in Natural and Artificial Systems*, The University of Michigan Press, Ann Harbor, MI, 1975.

[62] Dorigo, M., Maniezzo, V., and Colorni, A., Positive Feedback as a Search Strategy, Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.

[63] Moscato, P., On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, Report 826, California Institute of Technology, 1989.

2

Basic Methodologies and Applications

Teofilo F. Gonzalez
University of California, Santa Barbara

2.1 Introduction

In Chapter 1 we presented an overview of approximation algorithms and metaheuristics. This serves as an overview of Parts I, II, and III of this handbook. In this chapter we discuss in more detail the basic methodologies and apply them to simple problems. These methodologies are restriction, greedy methods, LP rounding (deterministic and randomized), α vector, local ratio and primal dual. We also discuss in more detail inapproximability and show that the "classical" version of the traveling salesperson problem (TSP) is constant ratio inapproximable. In the last three sections we present an overview of the application chapters in Parts IV, V, and VI of the handbook.

2.2 Restriction

Chapter 3 discusses *restriction* which is one of the most basic techniques to design approximation algorithms. The idea is to generate a solution to a given problem P by providing an optimal or suboptimal solution to a subproblem of P. A subproblem of a problem P means restricting the solution space for P by disallowing a subset of the feasible solutions. The idea is to restrict the solution space so that it has some structure, which can be exploited by an efficient algorithm that solves the problem optimally or suboptimally. For this approach to be effective the subproblem must have the property that, for every problem instance, its optimal or suboptimal solution has an objective function value that is "close" to the optimal one for P. The most common approach is to solve just one subproblem, but there are algorithms where more than one subproblem is solved and then the best of the solutions computed is the solution generated. Chapter 3 discusses this methodology and shows how to apply it to several problems. Approximation algorithms based on this approach are discussed in Chapters 35, 36, 42, 45, 46, 54, and 73. Let us now discuss a scheduling application in detail. This is the scheduling problem studied by Graham [1,2].

2.2.1 Scheduling

A set of n tasks denoted by $T_1, T_2, \ldots, T_n$ with processing time requirements $t_1, t_2, \ldots, t_n$ have to be processed by a set of m identical machines. A partial order C is defined over the set of tasks to enforce a set of precedence constraints or task dependencies. The partial order specifies that a machine cannot commence the processing of a task until all of its predecessors have been completed. Each task T_i has to be processed for t_i units of time by one of the machines. A (nonpreemptive) schedule is an assignment of tasks to time intervals on the machines in such a way that (1) each task T_i is processed continuously for t_i units of time by one of the machines; (2) each machine processes at most one task at a time; and (3) the precedence constraints are satisfied. The *makespan* of a schedule is the latest time at which a task is being processed. The scheduling problem discussed in this section is to construct a minimum makespan schedule for a set of partially ordered tasks to be processed by a set of identical machines. Several limited versions of this scheduling problem has been shown to be NP-hard [3].

Example 2.1

The number of tasks, n, is 8 and the number of machines, m, is 3. The processing time requirements for the tasks, and the precedence constraints are given in Figure 2.1, where a directed graph is used to represent the task dependencies. Vertices represent tasks and the directed edges represent task dependencies. The integers next to the vertices represent the task processing requirements. Figure 2.2 depicts two schedules for this problem instance.

In the next subsection, we present a simple algorithm based on restriction to generate provable good solutions to this scheduling problem. The solution space is restricted to schedules without forced "idle time," i.e., each feasible schedule does not have idle time from the time at which all the predecessors of task T_i (in C) are completed to the time when the processing of task T_i begins, for each i.

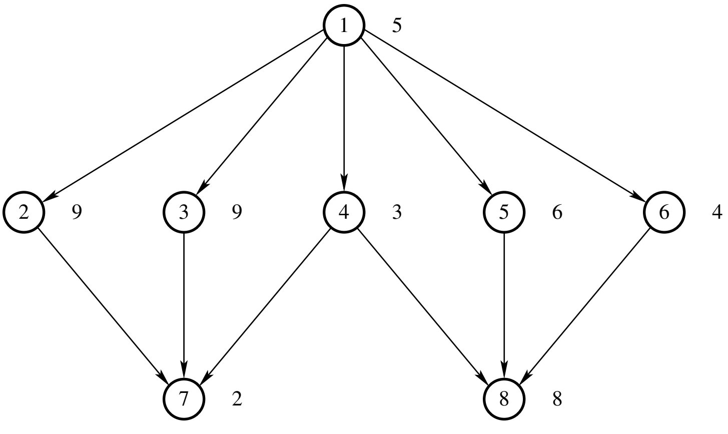


FIGURE 2.1 Precedence constraints and processing time requirements for Example 2.1.

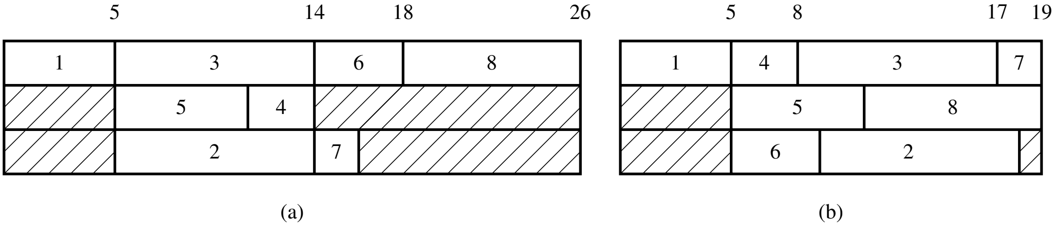


FIGURE 2.2 (a) and (b) represent two different AAT schedules for Example 2.1. Schedule (b) is a minimum makespan schedule.

2.2.2 Partially Ordered Tasks

Let us further restrict the scheduling policy to construct a schedule from time zero till all the tasks have been assigned. The scheduling policy is: whenever a machine becomes idle we assign one of the unassigned tasks that is ready to commence execution, i.e., we have completed all its predecessors. Any scheduling policy in this category can be referred to as a *no-additional-delay* scheduling policy. The simplest version of this scheduling policy is to assign any of the tasks (AAT) ready to be processed. A schedule generated by this policy is called an AAT schedule. These schedules are like the list schedules [1] discussed in Chapter 1. The difference is that list schedules have an ordered list of tasks, which is used to break ties. The analysis for both types of algorithms is the same since the list could be any list.

In Figure 2.2 we give two possible AAT schedules. The two schedules were obtained by breaking ties differently. The schedule in Figure 2.2(b) is a minimum makespan schedule. The reason for this is that the machines can only process one of the tasks T_1, T_5, or T_8 at a time, because of the precedence constraints.

Figure 2.2 suggests that an optimal schedule can be generated by just finding a clever method to break ties. Unfortunately, one cannot prove that this is always the case because there are problem instances for which all minimum makespan schedules are not AAT schedules.

The makespan of an AAT schedule is never greater than $2 - \frac{1}{m}$ times the one of an optimal schedule for the instance. This is expressed by

$$\frac{\hat{f}_I}{f_I^*} \leq 2 - \frac{1}{m}$$

where $\hat{f}_I$ is the makespan of any possible AAT schedule for problem instance I and f_I^* is the makespan of an optimal schedule for I. We establish this property in the following theorem:

Theorem 2.1

For every instance I of the identical machine scheduling problem, and every AAT schedule, $\frac{\hat{f}_I}{f_I^*} \leq 2 - \frac{1}{m}$.

Proof

Let S be any AAT schedule for problem instance I with makespan $\hat{f}_I$. By construction of the AAT schedules it cannot be that at some time $0 \leq t \leq \hat{f}_I$ all machines are idle. Let i_1 be the index of a task that finishes at time $\hat{f}_I$. For $j = 2, 3, \ldots,$ if task $T_{i_{j-1}}$ has at least one predecessor in C, then define i_j as the index of a task with latest finishing time that is a predecessor (in C) of task $T_{i_{j-1}}$. We call these tasks a *chain* and let k be the number of tasks in the chain. By the definition of task T_{i_j}, it cannot be that there is an idle machine from the time when task T_{i_j} completes its processing to the time when task $T_{i_{j-1}}$ begins processing. Therefore, a machine can only be idle when another machine is executing a task in the chain. From these two observations we know that

$$m\hat{f}_I \leq (m-1)\sum_{j=1}^{k} t_{i_j} + \sum_{j=1}^{n} t_j$$

Since no machine can process more than one task at a time, and since not two tasks, one of which precedes the other in C, can be processed concurrently, we know that an optimal makespan schedule satisfies

$$f_I^* \geq \frac{1}{m}\sum_{j=1}^{n} t_j \quad \text{and} \quad f_I^* \geq \sum_{j=1}^{k} t_{i_j}$$

Substituting in the above inequality, we know that $\frac{\hat{f}_I}{f_I^*} \leq 2 - \frac{1}{m}$. □

The natural question to ask is whether or not the approximation ratio $2 - \frac{1}{m}$ is the best possible for AAT schedules. The answer to this question is affirmative, and a problem instance for which this bound is tight is given in Example 2.2.

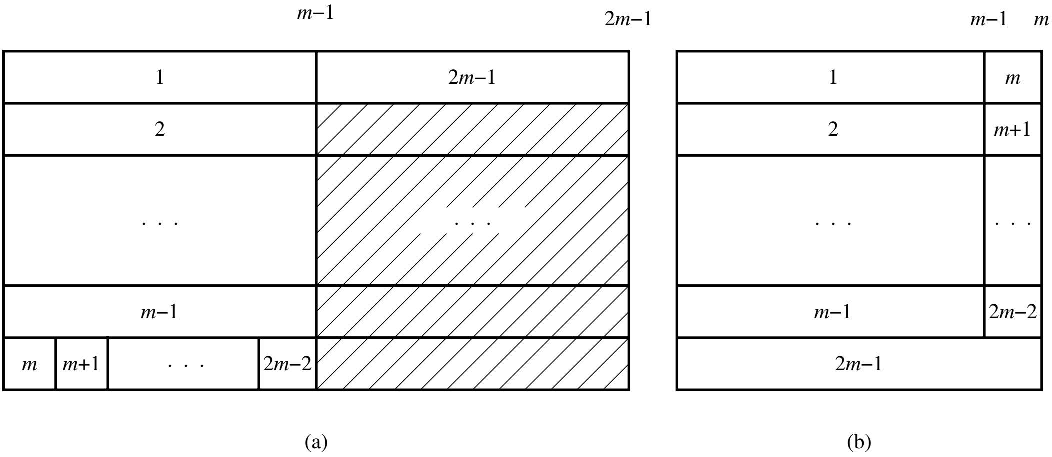


FIGURE 2.3 (a) AAT schedule. (b) Optimal schedule for Example 2.2.

Example 2.2

There are $2m - 1$ independent tasks. The first $m - 1$ tasks have processing time requirement $m - 1$, the next $m - 1$ tasks have processing time requirement one, and the last task has processing time requirement equal to m. An AAT schedule with makespan $2m - 1$ is given in Figure 2.3(a), and in Figure 2.3(b) we give a minimum makespan schedule.

Note that these results also hold for the list schedules [1] defined in Chapter 1. These type of schedules are generated by a no-additional-delay scheduling rule that is augmented by a list that is used to decide which of the ready-to-process tasks is the one to be assigned next.

Let us now consider the case when ties (among tasks that are ready) are broken in favor of the task with smallest index (T_i is selected before T_j if both tasks are ready to be processed and $i < j$). The problem instance I_A given in Figure 2.4 has three machines and eight tasks. Our scheduling procedure (augmented with a tie-breaking list) generates a schedule with makespan 14. In Chapter 1, we say that list schedules (which are this type of schedules) have anomalies. To verify this, apply the scheduling algorithm to instance I_A, but now there are four machines. One would expect a schedule for this new instance to have makespan at most 14, but you can easily verify that this is not the case. Now apply the scheduling algorithm to the instance I_A where every task has a processing requirement decreased by one unit. One would expect a schedule for this new instance to have makespan at most 14, but you can easily verify that is not the case. Apply the scheduling algorithm to the problem instance I_A without the precedence constraints from task

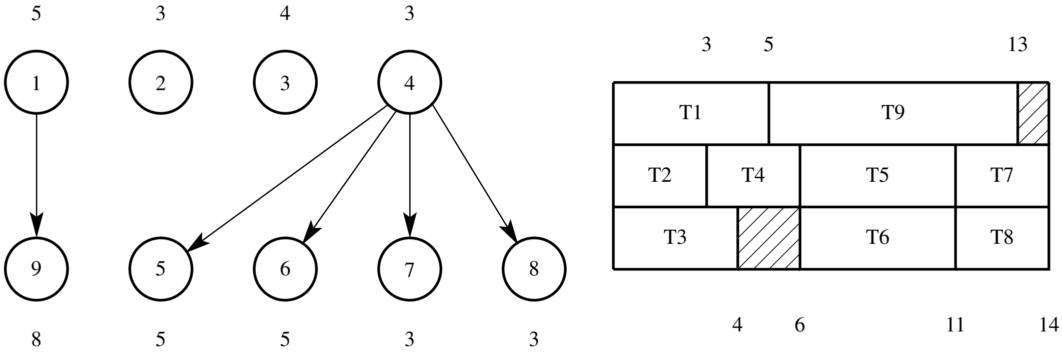


FIGURE 2.4 (a) Problem instance with anomalous behavior. (b) AAT schedule with tie-breaking list.

4 to 5 and task 4 to 6. One would expect a schedule for this instance to have makespan at most 14, but that is not the case. These are anomalies. Approximation algorithms suffer from this type of anomalous behavior. We need to be aware of this fact when using approximation algorithms.

As in the case of Example 2.2, the worst case behavior arises when the task with longest processing time is being processed while the rest of the machines are idle. Can a better approximation bound be established for the case when ties are broken in favor of a task with longest processing time (LPT)? The schedules generated by this rule are called *LPT schedules.* Any LPT schedule for the problem instance in Figure 2.3 is optimal. Unfortunately, this is not always the case and the approximation ratio in general is the same as the one for the AAT schedules. To see this just partition task $2m-1$ in Example 2.2 (see Figure 2.3[a]) into a two-task chain. The first one has processing requirement of ϵ, for some $0 < \epsilon < 1$, and the second one $m-\epsilon$. The schedule generated by the LPT rule will schedule first all the tasks with processing requirement greater than 1 and then the two tasks in the chain.

The problem with the LPT rule is that it only considers the processing requirements of the tasks ready to process, but ignores the processing requirements of the tasks that follow it. We define the *weight of a directed path* as the sum of the processing time requirements of the tasks in the path. Any directed path that starts at task t with maximum weight among all paths that start at task t is called a *critical path for task t.* The critical-path (CP) schedule is defined as a no-additional-delay schedule where the decision of which task to process next is a task whose CP weight is largest among the ready-to-be processed tasks. The CP schedule is optimal for the problem instance that was generated by replacing the last task in Example 2.2 by two tasks. However, Graham constructed problem instances for which the makespan of the CP schedule is $2 - 1/m$ times the length of an optimal schedule.

It is not known whether or not a polynomial-time algorithm exists with a smaller approximation ratio even when the processing time requirements for all the tasks are identical and $m \geq 3$. There is a polynomial-time algorithm that generates an optimal schedule when $m = 2$, but the problem with different task processing times is NP-hard. In the next subsection we present an algorithm with a smaller approximation ratio for scheduling independent task.

2.3 Greedy Methods

Another way to generate suboptimal solutions is to apply greedy algorithms. The idea is to generate a solution by making a sequence of irrevocable decisions. Each of these decisions is a best possible choice at that point, for example, select an edge of least weight, select the vertex of highest degree, or select the task with longest processing time. Chapter 4 discusses greedy methods. The discussion also includes primal-dual approximation algorithms falling into this category. Chapter 5 discusses the recursive greedy method. This methodology is for the case when making the best possible decision is an NP-hard problem. A large portion of the bin packing algorithms are greedy algorithms. Bin packing and its variants are discussed in Chapters 32–35. Other greedy methods appear in Chapters 36, 38, 39, 44–46, 49, 50, 58, 59, and 69. Let us now discuss the LPT scheduling rule for scheduling independent tasks on identical machines.

2.3.1 Independent Tasks

Another version of this scheduling problem that has received considerable attention is when the tasks are independent, i.e., the partial order between the tasks is empty. Graham's [2] elegant analysis for LPT scheduling has become a classic. In fact, the analysis of quite a few subsequent exact and approximation scheduling algorithms follow the same approach.

First, we analyze the LPT scheduling rule. For this case there is only one possible schedule, modulo the relabeling of the tasks. We call this a "greedy method" because of the ordering of the tasks with respect to their processing requirements. This tends to generate schedules where the shortest tasks end up being processed last and the resulting schedules tend to have near-optimal makespan. However as we shall see, one may obtain the same approximation ratio by just scheduling the tasks using a list where the $2m$ task with longest processing time appear first (in sorted order) and the remaining tasks appear next in any order. This approach could be called "limited greedy." We discuss other approximation algorithms for this problem after presenting the analysis for LPT schedules.

Let I be any problem instance with n independent tasks and m identical machines. We use $\hat{f}_I$, as the makespan for the LPT schedule for I and f_I^* as the one for an optimal schedule. In the next theorem we establish the approximation ratio for LPT schedules.

Theorem 2.2

For every scheduling problem instance I with n independent tasks and m identical machines, every LPT schedule satisfies $\frac{\hat{f}_I}{f_I^*} \leq \frac{4}{3} - \frac{1}{3m}$.

Proof

It is clear that LPT schedules are optimal for $m = 1$. Assume that $m \geq 2$. The proof is by contradiction. Suppose the above bound does not hold. Let I be a problem instance with the least number of tasks for which $\frac{\hat{f}_I}{f_I^*} > \frac{4}{3} - \frac{1}{3m}$. Let n the number of tasks in I, m the number of machines, and assume that $t_1 \geq t_2 \geq \cdots \geq t_n$. Let k be the smallest index of a task that finishes at time $\hat{f}_I$. It cannot be that $k < n$, as otherwise the problem instance $T_1, T_2, \ldots, T_k$ is also a counterexample and it has fewer tasks than instance I, but by assumption problem instance I is a counterexample with the least number of tasks. Therefore, k must be equal to n.

By the definition of LPT schedules, we know that there cannot be idle time before task T_n begins execution. Therefore,

$$\sum_{i=1}^{n} t_i + (m-1)t_n \geq m\hat{f}_I$$

This is equivalent to

$$\hat{f}_I \leq \frac{1}{m}\sum_{i=1}^{n} t_i + \left(1 - \frac{1}{m}\right)t_n$$

Since each machine cannot process more than one task at a time, we know that $f_I^* \geq \sum_{i=1}^{n} t_i/m$. Combining these two bounds we have

$$\frac{\hat{f}_I}{f_I^*} \leq 1 + \left(1 - \frac{1}{m}\right)\frac{t_n}{f_I^*}$$

Since I is a counterexample for the theorem, this bound must be greater than $\frac{4}{3} - \frac{1}{3m}$. Simplifying we know that $f_I^* < 3t_n$. Since t_n is the task with smallest processing time requirement it must be that in an optimal schedule, for instance, I none of the machines can process three or more tasks. Therefore, the number of tasks n is at most $2m$.

For problem instance I, let S^* be an optimal schedule with least $\sum f_i^2$, where f_i is the makespan in S^* for machine i. Assume without loss of generality that the tasks assigned to each machine are arranged from largest to smallest with respect to their processing times. All machines have at most two tasks, as S^* is an optimal schedule for I which by definition is a counterexample for the theorem.

Let i and j be two machines in schedule S^* such that $f_i > f_j$, machine i has two tasks and machine j has at least one task. Let a and b be the task index for the last task processed by machine i and j, respectively. It cannot be that $t_a > t_b$, as otherwise applying the interchange given in Figure 2.5(a) results

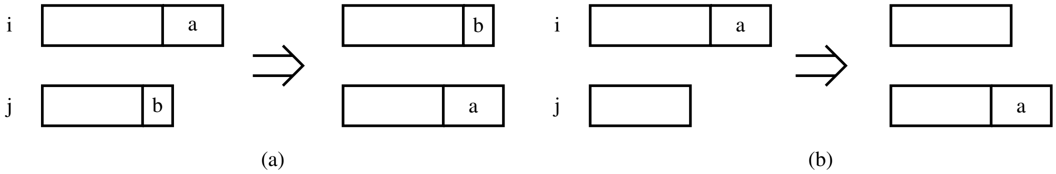


FIGURE 2.5 Schedule transformations.

in an optimal schedule with smaller $\sum f_i^2$. This contradicts the fact that S^* is an optimal schedule with least $\sum f_i^2$. Let i and j be two machines in schedule S^* such that machine i has two tasks. Let a be the task index for the last task processed by machine i. It cannot be that $f_i - t_a > f_j$ as otherwise applying the interchange given in Figure 2.5(b) results in an optimal schedule with smaller $\sum f_i^2$. This contradicts the fact that S^* is an optimal schedule with least $\sum f_i^2$.

Since the transformations given in Figure 2.5(a) and Figure 2.5(b) cannot apply, the schedule S^* must be of the form shown in Figure 2.6 after renaming the machines, i.e., machine i is assigned task T_i (if $i \leq n$) and task T_{2m-i+1} (if $2m - i + 1 \leq n$). But this schedule is an LPT schedule and $\hat{f} = f^*$. Therefore, there cannot be any counterexamples to the theorem. This completes the proof of the theorem. □

For all m there are problem instances for which the ratio given by Theorem 2.2 is tight. In Figure 2.7 we give one of such problem instance for three machines.

The important properties needed to prove Theorem 2.2 are that the longest $2m$ tasks need to be scheduled via LPT, and either the schedule will be optimal for the $2m$ task or at least three tasks will be assigned to a machine. The first set of m tasks, the ones with longest processing time, will be assigned to one machine each, so the order in which they are assigned is not really important. The next set of m tasks need to be assigned from longest to shortest processing times as in the LPT schedule. The remaining tasks can be assigned in any order as long as whenever a machine finishes a task the next task in the list is assigned to that machine. Any list schedule whose list follows the above ordering can be shown to have makespan at most $\frac{4}{3} - \frac{1}{3m}$ times the one of an optimal schedule. These type of schedules form a restriction on the solution space.

It is interesting to note that the problem of scheduling $2m$ independent tasks is an NP-hard problem. However, in polynomial time we can find out if there is an optimal schedule in which each machine has at most two tasks. And this is all that is needed to establish the $\frac{4}{3} - \frac{1}{3m}$ approximation ratio. One of the first

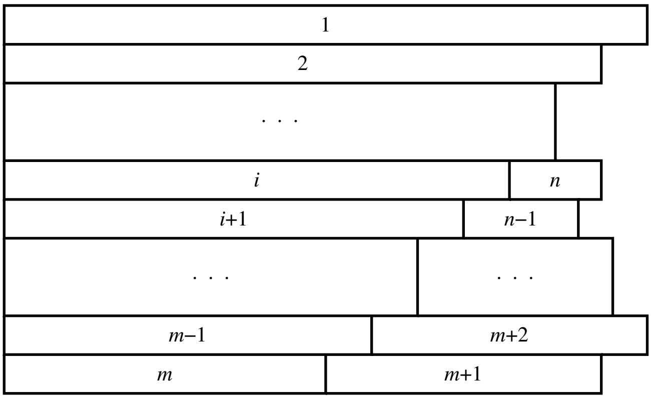
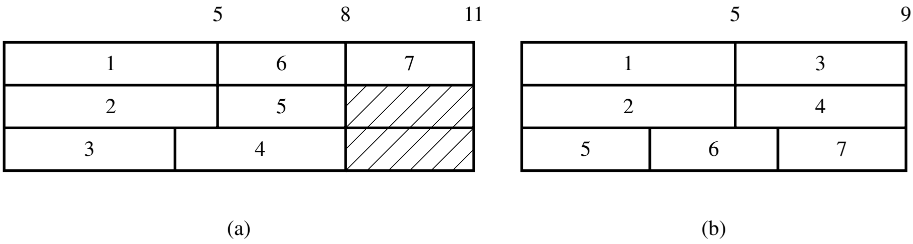


FIGURE 2.6 Optimal schedule.



(a)

(b)

FIGURE 2.7 (a) LPT schedule. (b) Optimal schedule.

avenues of research explored was to see if the same approach would hold for the longest $3m$ tasks. That is, give a polynomial-time algorithm that finds an optimal schedule in which each machine has at most three tasks. If such an algorithm exists, we could use it to generate schedules that are within $\frac{5}{4} - \frac{1}{4m}$ times the makespan of an optimal schedule. This does not seem possible as Garey and Johnson [3] established that this problem is NP-hard.

Other approximation algorithms with improved performance were subsequently developed. Coffman et al. [4] introduced the multifit (MF) approach. A k attempt MF approach is denoted by MF_k. The MF_k procedure performs k binary search steps to find the smallest capacity c such that all the tasks can be packed into a set of m bins when packing using first fit with the tasks sorted in nondecreasing order of their processing times. The tasks assigned to bin i correspond to machine i and c is the makespan of the schedule. The approximation ratio has been shown to be $1.22 + 2^{-k}$ and the time complexity of the algorithm is $O(n \log n + kn \log m)$. Subsequent improvements to $1.2 + 2^{-k}$ [5] and $\frac{72}{61} + \frac{1}{2^k}$ [6] were possible within the same time complexity bound. However, the latter algorithm has a very large constant associated with the big "oh" bound.

Following a suggestion by D. Kleitman and D. E. Knuth, Graham [2] was led to consider the following scheduling strategy. For any $k \geq 0$ an optimal schedule for the longest k tasks is constructed and then the remaining tasks are scheduled in any order using the no-additional-delay policy. He shows that this algorithm has an approximation ratio $1 + \frac{1-1/m}{1+\lceil k/m \rceil}$ and takes $O(n \log m + km^k)$ time when there is a fixed number of machines. This was the first polynomial-time approximation scheme for any problem. This polynomial-time approximation scheme, as well as the ones for other problems are explained in more detail in Chapter 9. Fully polynomial-time approximation schemes are not possible for this problem unless $P = NP$ [3].

2.4 Relaxation: Linear Programming Approach

Let us now consider the minimum-weight vertex cover, which is a fundamental problem in the study of approximation algorithms. This problem is defined as follows

Problem: Minimum-weight vertex cover.

Instance: Given a vertex weighted undirected graph G with the set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, edges $E = \{e_1, e_2, \ldots, e_m\}$ and a positive real number (weight) w_i assigned to each vertex v_i.

Objective: Find a minimum-weight vertex cover, i.e., a subset of vertices $C \subset V$ such that every edge is incident to at least one vertex in C. The weight of the vertex cover C is the sum of the weight of the vertices in C.

It is well known that the minimum-weight vertex cover problem is an NP-hard problem. Now consider the following simple greedy algorithm to generate a vertex cover. Assume without loss of generality that the graph G does not have isolated vertices, i.e., vertices without any edges. An edge is said to be *uncovered* with respect to a set of vertices C if both of its endpoints are vertices in $V \backslash C$, i.e., if both endpoints are not in C.

```
Algorithm Min-Weight($G$)
    Let $C = \emptyset$;
    while there is an uncovered edge do
        Let $U$ be the set of vertices adjacent to at least one uncovered edge;
        Add to $C$ a least weight vertex in set $U$;
    endwhile
end
```

Algorithm `Min-Weight` is not a constant-ratio approximation algorithm for the vertex cover problem. Consider the family of star graphs $\mathcal{K}$ each with $l + 1$ nodes, l edges, the center vertex having weight k and the l leaves having weight 1, for any positive integers $k \geq 2$ and $l \geq 3$. For each of these graphs Algorithm `Min-Weight` generates a vertex cover that includes all the leaves in the graph and the weight of the cover is l. For all graphs in $\mathcal{K}$ with $k = 2$, an optimal cover has weight 2 and includes only the center vertex. Therefore, Algorithm `Min-Weight` has an approximation ratio of at least $l/2$, which cannot be bounded above by any fixed constant.

Algorithm `Max-Weight` is identical to Algorithm `Min-Weight`, but instead of selecting the vertex in set U with least weight, it selects one with largest weight. Clearly, this algorithm constructs an optimal cover for the graphs identified above where Algorithm `Min-Weight` performs badly. For every graph in $\mathcal{K}$, this algorithm selects as its vertex cover the center vertex which has weight k. Now for all graphs in $\mathcal{K}$ with $l = 2$, an optimal cover consists of both leaf vertices and it has weight 2. Therefore, the approximation ratio for Algorithm `Max-Weight` is at least $k/2$, which cannot be bounded above by any fixed constant.

All of the graphs identified above, where one of the algorithms performs badly, have the property that the other algorithm constructs an optimal solution. A compound algorithm that runs both algorithms and then selects the better of the two vertex covers may be a constant-ratio algorithm for the vertex cover problem. However, this compound algorithm can also be easily fooled by just using a graph consisting of two stars, where each of the individual algorithms failed to produce good solutions. Therefore, this compound algorithm fails to generate constant-ratio approximate solutions. One may now argue that we could partition the graph into connected components and apply both algorithms to each component. For these "two-star" graphs the new compound algorithm will generate an optimal solution. But in general this new approach fails to produce a constant-ratio approximate solution for all possible graphs. Adding an edge between the center vertex in the "two-star" graphs gives rise to problem instances for which the new compound algorithm fails to provide a constant ratio approximate solution.

A more clever approach is a modified version of Algorithm `Min-Weight`, where instead of selecting a vertex of least possible weight in set U, one selects a vertex v in set U with least $w(v)/u(v)$, where $u(v)$ is the number of uncovered edges incident to vertex v. This seems to be a better strategy because when vertex v is added to C it covers $u(v)$ edges at a total cost of $w(v)$. So the cost (weight) per edge of $w(v)/u(v)$ is incurred when covering the uncovered edges incident to vertex v. This strategy solves optimally the star graphs in $\mathcal{K}$ defined above. However, even when all the weights are equal, one can show that this is not a constant ratio approximation algorithm for the weighted vertex cover problem. In fact, the approximation ratio for this algorithm is about $\log n$. Instances with a simple recursive structure that asymptotically achieve this bound as the number of vertices increases can be easily constructed. Chapter 3 gives an example of how to construct problem instances where an approximation algorithm fails to produce a good solution.

Other approaches to solve the problem can also be shown to fail to provide a constant-ratio approximation algorithm for the weighted vertex cover. What type of algorithm can be used to guarantee a constant-ratio solution to this problem? Let us try another approach.

Another way to view the minimum-weight vertex cover is by defining a 0/1 variable x_i for each vertex v_i in the graph. The 0/1 vector X defines a subset of vertices C as follows. Vertex v_i is in C if and only if $x_i = 1$. The set of vertices C defined by X is a vertex cover if and only if for every edge $\{i, j\}$ in the graph $x_i + x_j \geq 1$. The vertex cover problem is expressed as an instance of the 0/1 integer linear programming (ILP) as follows:

$$\text{minimize} \quad \sum\nolimits_{i \in V} w_i x_i \tag{2.1}$$

$$\text{subject to} \quad x_i + x_j \geq 1 \;\; \forall \{i, j\} \in E \tag{2.2}$$

$$x_i \in \{0, 1\} \quad \forall i \in V \tag{2.3}$$

The 0/1 ILP is also an NP-hard problem.

An important methodology for designing approximation algorithms is relaxation. In this case one relaxes the integer constraints for the x_i values. That is, we replace constraint (2.3) ($x_i = \{0, 1\}$) by $0 \leq x_i \leq 1$ (or simply $x_i \geq 0$, which in this case is equivalent). This means that we are augmenting the solution space by adding solutions that are not feasible for the original problem. This approach will at least provide us with what appears to be a good lower bound for the value of an optimal solution of the original problem, since every feasible solution to the original problem is a feasible solution to the relaxed problem (but the converse is not true). This relaxed problem is an instance of the linear programming (LP) problem which can be solved in polynomial time. Let X^* be an optimal solution to the LP problem. Clearly, X^* might not be a vertex cover as the x_i^* values may be noninteger. The previous interpretation for the X^* values has been lost because it does not make sense to talk about a fractional part of a vertex being part of a vertex cover. To circumvent this situation, we need to use the X^* vector to construct a 0/1 vector $\hat{X}$ that represents a vertex cover. For a vector $\hat{X}$ to represent a vertex cover it needs to satisfy inequality (2.2) (i.e., $\hat{x}_i + \hat{x}_j \geq 1$), for every edge $e_k = \{i, j\} \in E$. Clearly, the inequalities hold for X^*. This means that for each edge $e_k = \{i, j\} \in E$ at least one of x_i^* or x_j^* has value at least greater than or equal to $\frac{1}{2}$. So the vector $\hat{X}$ defined from X^* as $\hat{x}_i = 1$ if $x_i^* \geq \frac{1}{2}$ (rounding up) and $\hat{x}_i = 0$ if $x_i^* < \frac{1}{2}$ (rounding down) represents a vertex cover. Furthermore, because of the rounding up the objective function value for the vertex cover $\hat{X}$ is at most $2 \sum w_i x_i^*$. Since $\sum w_i x_i^*$ value is a lower bound for an optimal solution to the weighted vertex cover problem, we know that this procedure generates a vertex cover whose weight is at most twice the weight of an optimal cover, i.e., it is a 2-approximation algorithm. This process is called (deterministic) *LP rounding*. Chapters 6, 7, 9, 11, 37, 45, 57, 58, and 70 discuss and apply this methodology to other problems.

Another way to round is via randomization, which means in this case that we flip a biased coin (with respect to x_i^* and perhaps other factors) to decide the value for $\hat{x}_i$. The probability of $\hat{X}$ is a vertex cover and its expected weight can be computed. By repeating this randomization process several times, one can show that a cover with weight at most twice the optimal one will be generated with very high probability. In this case it is clear that randomization is not needed. However, for other problems it is justified. Chapters 4, 6, 7, 11, 12, 57, 70, and 80 discuss LP randomized rounding, and Chapter 8 discusses more complex randomized rounding for semidefinite programming (SDP).

The above rounding methods have the disadvantage that an LP problem needs to be solved. Experimental evaluations over several decades have shown that the Simplex method solves quickly (in poly time) the LP problem. But the worst-case time complexity is exponential with respect to the problem size. In Chapter 1 we have discussed the Ellipsoid algorithm and more recent ones that solve LP problems. Even though these algorithms have polynomial-time complexity, there is a term that depends on the number of bits needed to represent the input. Much progress has been made in speeding up these procedures, but the algorithms are not competitive with typical $O(n^2)$ time algorithms for other problems.

Let us now discuss another approximation algorithm for the minimum vertex cover problem that it is "independent" of LP, and then we discuss a local-ratio and a primal-dual approach to this problem.

We call this approach the *α-vector* approach. For every vertex $i \in V$, we define $\delta(i)$ as the set of edges incident to vertex i. Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ be any vector of m nonnegative real values, where $m = |E|$ is the number of edges in the graph. For all k multiply the kth edge inequality by α_k,

$$\alpha_k x_i + \alpha_k x_j \geq \alpha_k \quad \forall e_k = \{i, j\} \in E \tag{2.4}$$

The total sum of these inequalities can be expressed as

$$\sum_{i \in V} \sum_{e_k \in \delta(i)} \alpha_k x_i \geq \sum_{e_k \in E} \alpha_k \tag{2.5}$$

Define $\beta_i = \sum_{e_k \in \delta(i)} \alpha_k$ for every vertex $i \in V$. In other words, β_i be the sum of the α values of all the edges incident to vertex i. Substituting in the above inequality we know that

$$\sum_{i \in V} \beta_i x_i \geq \sum_{e_k \in E} \alpha_k \tag{2.6}$$

Suppose that the α vector is such that $w_i \geq \beta_i$ for all i. Then it follows that

$$\sum_{i \in V} w_i x_i \geq \sum_{i \in V} \beta_i x_i \geq \sum_{e_k \in E} \alpha_k \tag{2.7}$$

In other words any vector α such that the resulting vector β computed from it satisfies $w_i \geq \beta_i$ provides us with the lower bound $\sum_{e_k \in E} \alpha_k$ for the objective function value of every vector X that represents a vertex cover. In other words, if we assign a positive weight to each edge in such a way that the sum of the weight of the edges incident to each vertex i is at most w_i, then the sum of the weight of the edges is a lower bound for an optimum solution.

This is a powerful lower bound. To get maximum strength we need to find a vector α such that $\sum_{e_k \in E} \alpha_k$ is maximum. But finding this vector is as hard as solving the LP problem described earlier. What if we find a maximal vector α, i.e., a vector that cannot possibly be increased in any of its components? This is a simple task. It is just a matter of starting with an α vector with all entries being zero and then increasing one of its components until it is no longer possible to do so. We keep on doing this until there are no edges whose α value can be increased. In this maximal solution, we know that for each edge in the graph at least one of its endpoints has the property that $\beta_i = w_i$, as otherwise the maximality of α is contradicted. Define the vector $\hat{X}$ from the α vector as follows: $x_i = 1$ if $\beta_i = w_i$, and $x_i = 0$, otherwise. Clearly, $\hat{X}$ represents a vertex cover because for every edge in the graph we know that at least one of its vertices has $\beta_i = w_i$. What is the weight of the vertex cover represented by $\hat{X}$? We know that $\sum w_i \hat{x}_i = \sum \beta_i \hat{x}_i \leq 2 \sum \alpha_k$ because each α_k can contribute its value to at most two β_is. Therefore, we have a simple 2-approximation algorithm for the weighted vertex cover problem. Furthermore, the procedure to construct the vertex cover takes linear time with respect to the number of vertices and edges in the graph.

This algorithm was initially developed by Bar-Yehuda and Even [7] using the LP relaxation and its dual. This approach is called the *primal-dual* approach. It will be discussed later in this section. The above algorithm can be proven to be a 2-approximation algorithm without using the ILP formulation. That is, the same result can be established by just using simple combinatorial arguments [8].

Another related approach, called *local ratio*, was developed by Bar-Yehuda and Even [9]. Initially, each vertex is assigned a cost which is simply its weight and it is referred to as the *remaining cost*. At each step the algorithm makes a "down payment" on a pair of vertices. This has the effect of decreasing the remaining cost of each of the two vertices. Label the edges in the graph $\{e_1, e_2, \ldots, e_m\}$. The algorithm considers one edge at a time using this ordering. When the kth edge $e_k = \{i, j\}$ is considered, define γ_k as the minimum of the remaining cost of vertex i and vertex j. The edge makes a down payment of γ_k to each of its two endpoints and each of the two vertices has its remaining cost decreased by γ_k. The procedure stops when we have considered all the edges. All the vertices whose current cost is zero have been paid for completely and they are yours to keep. The remaining ones have not been paid for and there are "no refunds" (not even if you talk to the store manager). The vertices that have been paid for completely form a vertex cover. The weight of all the vertices in the cover generated by the procedure is at most twice $\sum_{e_k \in E} \gamma_k$, which is simply the sum of the down payments made. What is the weight of an optimal vertex cover? The claim is it is equal to $\sum_{e_k \in E} \gamma_k$. The reason is simple. Consider the first step when we introduce γ_1 for edge e_1. Let I_0 be the initial problem instance and I_1 be the resulting instance after deleting edge e_1 and reducing the cost of the two endpoints of edge e_1 by γ_1. One can prove that $f^*(I_0) = f^*(I_1) + \gamma_1$, and inductively that $f^*(I_0) = \sum_{e_k \in E} \gamma_k$ [10]. The algorithm is a 2-approximation algorithm for the weighted vertex cover. The approach is called *local ratio* because at each step one adds $2\gamma_k$ to the value of the solution generated and one accounts for γ_k value of an optimal solution. This local-ratio approach has been successfully applied to quite a few problems. The best feature of this approach is that it is very simple to understand and does not require any LP background.

The primal-dual approach is similar to the previous ones, but it uses the foundations of LP theory. The LP relaxation problem is

$$\text{minimize} \quad \sum_{i \in V} w_i x_i \tag{2.8}$$

$$\text{subject to} \quad x_i + x_j \geq 1 \quad \forall e_k = \{i, j\} \in E \tag{2.9}$$

$$x_i \geq 0 \qquad \forall i \in V. \tag{2.10}$$

The LP problem is called the *primal* problem. The corresponding dual problem is

$$\text{maximize} \quad \sum_{e_k \in E} y_k \tag{2.11}$$

$$\text{subject to} \quad \sum_{e_k \in \delta(i)} y_k \leq w_i \quad \forall i \in V \tag{2.12}$$

$$y_k \geq 0 \qquad \forall e_k \in E \tag{2.13}$$

As you can see the Y vector is simply the α vector defined earlier, and the dual is to find a Y vector with maximum $\sum_{i \in V} y_i$. Linear programming theory [11,12] states that any feasible solution X to the primal problem and any feasible solution Y to the dual problem are such that

$$\sum_{e_k \in E} y_k \leq \sum_{i \in V} w_i x_i$$

This is called *weak duality. Strong duality* states that

$$\sum_{e_k \in E} y_i^* = \sum_{i \in V} w_i x_i^*$$

where X^* is an optimal solution to the primal problem and Y^* is an optimal solution to the dual problem. Note that the dual variables are multiplied by weights which are the right-hand side of the constraints in the primal problem. In this case all of them are one.

The primal-dual approach is based on the weak duality property. The idea is to first construct a feasible solution to the dual problem. That solution will give us a lower bound for the value of an optimal vertex cover, in this case. Then we use this solution to construct a solution to the primal problem. The idea is that the difference of the objective function value between the primal and dual solutions we constructed is "small." In this case we construct a maximal vector Y (as we did with the α vector before). Then we note that since the Y vector is maximal, then for at least one of the endpoints (say i) of every edge must satisfy Inequality 2.12 tight, i.e., $\sum_{e_k \in \delta(i)} y_k = w_i$. Now define vector X with $x_i = 1$ if inequality (2.12) is tight in the dual solution. Clearly, X represents a feasible solution to the primal problem and its objective function value is at most $2 \sum_k y_k$. It then follows by weak duality that an optimal weighted vertex cover has value at least $\sum_k y_k$ and we have a 2-approximation algorithm for the weighted vertex cover. It is simple to see that the algorithm takes linear time (with respect to the number of vertices and edges in the graph) to solve the problem.

There are other ways to construct a solution to the dual problem. In Chapters 4 and 13 another method is discussed for finding a solution to the dual problem. Note the difference in the time required to construct the solution. Chapter 13 discusses a "distributed" version of this algorithm. This algorithm makes decisions by using only "local" information. Chapters 37, 39, 40, and 71 discuss several approximation algorithms based on variations of the primal dual approach. Some of these methods are not exactly primal dual, but may be viewed this way.

Linear programming has also been used as a tool to compute the approximation ratio of some algorithms. This type of research may eventually be called the *automatic analysis of approximation algorithms.* Chapter 3 discusses an early approach to compute the approximation ratio, and Chapter 39 discusses a more recent one. In the former case, a set of LP needed to be solved. Once this was computed it gave the necessary insight on how to prove it analytically. In the latter case, one just formulates the problem and finds bounds for the value of an optimal solution to the LP problem.

2.5 Inapproximability

Sahni and Gonzalez [13] established that constant-ratio polynomial time approximation algorithms exist for some problems only if $P = NP$. In other words, finding a suboptimal solution to some problems is as hard as finding an optimal solution. Any polynomial-time algorithm that generates k-approximate solution can be used to find an optimal solution to the problem in polynomial-time. One of these problems is the "classical" version of the TSP defined in Chapter 1, not the restricted one defined over metric graphs. To prove this result we show that an NP-complete problem, called the Hamiltonian Cycle (HC) problem, can be solved in polynomial time if there is a polynomial-time algorithm for the TSP that generates a k-approximate solution, for any fixed constant k. The HC problem is given an undirected graph, $G = (V, E)$, determine whether on not the graph has a HC. A HC for an undirected graph G is a path that starts at vertex 1, visits each vertex *exactly* once, and ends at vertex 1.

To prove this result a polynomial transformation (Chapter 1 and [3]) is used. Let $G = (V, E)$ be any instance of the HC problem with $n = |V|$. Now construct an instance $G' = (V', E', W')$ of the TSP as follows. The graph G' has n vertices and it is complete (all the edges are present). The edge $\{i, j\}$ in E' has weight 1 if the edge $\{i, j\}$ is in E, and weight Z otherwise. The value of Z is $(k-1)n + 2 > 1$. It will be clear later on why it was defined this way. If the graph G has a HC, then we know that the graph G' has a tour with weight n. However, if G does not have a HC, then all tours for the graph G' have weight greater than or equal to $n - 1 + Z$. A k-approximate solution (tour) when $f^*(G') = n$ must have weight at most $\hat{f}(G') \leq kf^*(G') = kn$. When G does not have a HC, the best possible tour that can be found by the approximation algorithm is one with weight at least $n - 1 + Z = kn + 1$. Therefore, if the approximation algorithm returns a tour with weight at most kn, then G has a HC, otherwise (the tour returned has weight $> kn$) G does not have a HC. Since the algorithm takes polynomial-time with respect to the number of vertices and edges in the graph, it then follows that the algorithm solves in polynomial time the HC problem. So we say that the TSP is inapproximable with respect to any constant ratio. It is inapproximable in the sense that a polynomial-time constant-ratio approximation algorithm implies the solution to a computational complexity question. In this case it is the $P = NP$ question.

In the last 15 years there have been new inapproximability results. These results have been for constant, $\ln n$, and n^ϵ approximation ratios. The techniques to establish some of these results are quite complex, but an important component continues to be reducibility. Chapter 17 discusses all of this work in detail.

2.6 Traditional Applications

We have used the label “traditional applications” to refer to the more established combinatorial optimization problems. Although some of the problems falling into the other categories also fall into this category and vice versa. The problems studied in this part of the handbook fall into the following categories: bin packing, packing, facility dispersion and location, traveling salesperson, Steiner tree, scheduling, planning, generalized assignment, and satisfiability. Let us briefly discuss these categories.

One of the fundamental problems in approximations is the bin packing problem. Chapter 32 discusses online and offline algorithms for one-dimensional bin packing. Chapters 33 and 34 discuss variants of the bin packing problem. This include variations that fall into the following type of problems: the number of items packed is maximized while keeping the number of bins fixed; there is a bound on the number of items that can be packed in each bin; dynamic bin packing, where each item has an arrival and departure time; the item sizes are not known, but the ordering of the weights is known; items may be fragmented while packing them into fixed capacity bins, but certain items cannot be assigned to the same bin; bin stretching; variable sized bin packing problem; and the bin covering problem.

Chapter 35 discusses several ways to generalize the bin packing problem to more dimensions. Two- and three-dimensional strip packing, bin packing in dimensions two and higher, vector packing, and several other variations are discussed. Primal-dual approximation algorithms for packing and stabbing (or covering) problems are covered in Chapter 37. Cutting and packing problems with important applications in the wood, glass, steel, and leather industries as well as in very large-scale integration (VLSI) design, newspaper paging, and container and truck loading are discussed in Chapter 36. For several decades, cutting and packing has attracted the attention of researchers in various areas including operations research, computer science, manufacturing, etc.

Facility dispersion problems are covered in Chapter 38. Dispersion problems arise in a number of applications, such as locating obnoxious facilities, choosing sites for business franchises, and selecting dissimilar solutions in multiobjective optimization. The facility location problem that model the placement of “desirable” facilities such as warehouses, hospitals, and fire stations are discussed in Chapter 39. These algorithms are called “dual fitting and factor revealing.”

Very interesting approximation algorithms for the prize collecting TSP is studied in Chapter 40. In this problem a salesperson has to collect a certain amount of prizes (the quota) by visiting cities. A known prize can be collected in every city. Chapter 41 discusses branch-and-bound algorithms for the TSP. These algorithms have been implemented to run in a multicomputer environment. A general software tool for running branch and bound algorithms in a distributed environment is discussed. This framework may be used for almost any divide-and-conquer computation. With minor adjustments, this tool can take any algorithm defined as a computation over directed acyclic graph, where the nodes refer to computations and the edges specify a precedence relation between computations, and run in a distributed environment.

Approximation algorithms for the Steiner tree problem are discussed in Chapter 42. This problem has applications in several research areas. One of these areas is VLSI physical design. In Chapter 43, practical approximations for a restricted Steiner tree problem are discussed.

Meeting deadline constraints is of great importance in real-time systems. In situations when this is not possible, it is often more desirable to execute some parts of every task, than to give up completely the execution of some tasks. This model allows for the trade-off of the quality of computations in favor of meeting the deadline constraints. Every task is logically decomposed into two subtasks, mandatory and optional. This type of scheduling problems fall under the imprecise computation model. These problems are discussed in Chapter 44. Chapter 45 discussed approximation algorithms for the *malleable task* scheduling problem. In this model, the processing time of a task depends on the number of processors allotted to it. A generalization of both the bin packing and TSP is the vehicle scheduling problem. Approximation algorithms for this problem are discussed in Chapter 46.

Automated planning consists of finding a sequence of actions that transforms an initial state into one of the goal states. Planning is widely applicable, and has been used in such diverse application domains as spacecraft control, planetary rover operations, automated nursing aides, image processing, computer security, and automated manufacturing. Chapter 47 discusses approximation algorithms and heuristics for problems falling into this category.

Chapter 48 presents heuristics and metaheuristics for the generalized assignment problem. This problem is a natural generalization of combinatorial optimization problems including bipartite matching, knapsack and bin packing problems; and has many important applications in flexible manufacturing systems, facility location, and vehicle routing problems.

Chapter 49 examines probabilistic greedy heuristics for maximization and minimization versions of the satisfiability problem.

2.7 Computational Geometry and Graph Applications

The problems falling into this category have applications in several fields of study, but can be viewed as computational geometry and graph problems. The problems studied in this part of the handbook fall into the following categories: 2D and 3D triangulations, connectivity problems, design and evaluation of geometric networks, pair decompositions, minimum edge length partitions, digital geometry, disjoint path problems, graph partitioning, graph coloring, finding subgraphs or trees with certain properties, etc.

Triangulation is not only an interesting theoretical problem in computational geometry, it also has many important applications, such as finite element methods for computer-aided design (CAD) and physical simulations. Chapter 50 discusses approximation algorithms for triangulations in two and three dimensions.

Chapter 51 examines approximation schemes for various geometric minimum-cost k-connectivity problems and for geometric survivability problems, giving a detailed tutorial of the novel techniques developed for these algorithms.

Geometric networks arise in many applications. Road networks, railway networks, telecommunication, pattern matching, bioinformatics—any collection of objects in space that have some connections between them can be modeled as a geometric network. Chapter 52 considers the problem of designing a "good" network and the dual problem, i.e., evaluating how "good" a given network is. Chapter 53 gives an overview of several proximity problems that can be solved efficiently using the well-separated pair decomposition (WSPD). A WSPD may be regarded as a "small" set of edges that approximates the dense complete Euclidean graph.

Approximation algorithms for minimum edge length partitions of rectangles with interior points are discussed in Chapter 54. This problem has applications in the area of CAD of integrated circuits and systems. Chapter 55 considers partitions of finite d-dimensional integer grids by lines in two-dimensional space or by hyperplanes and hypersurfaces in an arbitrary dimension. Some of these problems arise in the areas of digital image processing (analysis) and neural networks. Chapter 56 discusses the problem of finding a planar subgraph of maximum weight in a given graph. Problems of this form have applications in circuit layout, facility layout, and graph drawing.

Finding disjoint paths in graphs is a problem that has attracted considerable attention from at least three perspectives: graph theory, VLSI design, and network routing/flow. The corresponding literature is extensive. Chapter 57 explores offline approximation algorithms for problems on general graphs as influenced from the network flow perspective.

Chapter 58 surveys approximation algorithms and hardness results for different versions of the generalized Steiner network problem in which we seek to find a low-cost subgraph that satisfies prescribed connectivity requirements. These problems include the following well-known problems: min-cost k-flow, min-cost spanning tree, traveling salesman, directed/undirected Steiner tree, Steiner forest, k-edge/node-connected spanning subgraph, and others.

Besides numerous network design applications, spanning trees also play an important role in several newly established research areas, such as biological sequence alignments and evolutionary tree construction. Chapter 59 explores the problem of designing approximation algorithms for spanning-tree problems under different objective functions. It focuses on approximation algorithms for constructing efficient communication spanning trees.

Graph partitioning problem arises in a wide range of applications. Due to the complexity of the problem, heuristics have to be applied to partition large graphs in a reasonable amount of time. Chapter 60 discusses different approaches to the graph partitioning problem. The k-way partitioning of a hypergraph problem seeks to minimize a given cost function of such an assignment. A standard cost function is *net cut*, which is the number of hyperedges that span more than one partition, or, more generally, the sum of weight of such edges. Constraints are typically imposed on the solution, and make the problem difficult. Several heuristics for this problem are discussed in Chapter 61.

In many applications such as design of transportation networks, one often needs to identify a set of regions/sections whose damage will cause the greatest increase in transportation cost within the network. Once identified, extra protection can be deployed to prevent them from being damaged. A version of this problem is finding the most vital edges whose removal will cause the greatest damage to a particular property of the graph. The problems are traditionally referred to as prior analysis problems in sensitivity analysis and it is discussed in Chapter 62.

Stochastic local search algorithms for the classical graph coloring problem are discussed in Chapter 63. This problem arises in many real-life applications like register allocation, air traffic flow management, frequency assignment, light wavelengths assignment in optical networks, or timetabling. Chapter 64 discusses ant colony optimization (ACO) for solving the maximum disjoint paths problems. This problem has many applications including the establishment of routes for connection requests between physically separated network endpoints.

2.8 Large-Scale and Emerging Applications

The problems arising in the areas of wireless and sensor networks, multicasting, multimedia, bioinformatics VLSI CAD, game theory, data analysis, digital reputation, and color quantization may be referred to as problems in "emerging" applications and normally involve large-scale problems instances. Some of these problems also fall in the other application areas.

Chapter 65 describes existing multicast routing protocols for ad hoc and sensor networks, and analyze the issue of computing minimum cost multicast trees. The multicast routing problem, and approximation algorithms for mobile ad hoc networks (MANETs) and wireless sensor networks (WSNs) are presented.

Since flat networks do not scale, it is important to overlay a virtual infrastructure on a physical network. The design of the virtual infrastructure should be general enough so that it can be leveraged by a multitude of different protocols. Chapter 66 proposes a novel clustering scheme based on a number of properties of diameter-2 graphs. Extensive simulation results have shown the effectiveness of the clustering scheme when compared to other schemes proposed in the literature.

Ad hoc networks are formed by collections of nodes which communicate with each other through radio propagation. Topology control problems in such networks deal with the assignment of power values to the nodes so that the power assignment leads to a graph topology satisfying some specified properties. The problem is to minimize a specified function of the powers assigned to the nodes. Chapter 67 discusses some known approximation algorithms for this type of problems. The focus is on approximation algorithms with proven performance guarantees.

An important requirement of wireless ad hoc networks is that they should be self-organizing. Energy conservation and network performance are probably the most critical issues in wireless ad hoc networks, because wireless devices are usually powered by batteries only and have limited computing capability and memory. Many proposed methods apply computational geometry technique (specifically, geometrical spanner) to achieve power efficiency. In Chapter 68, approximation algorithms of power spanner for ad hoc networks are reviewed.

As networks continue to grow explosively both in size and internal complexity, the ever-increasing tremendous traffic load and applications drive researchers to develop techniques for analyzing network performance and managing network resources. To accomplish this, one needs to know the current internal structure of the network. Discovery of internal information such as topology and localized lossy links plays an important role in resource management, loss recovery, and congestion control. Chapter 69 proposes a way to identify this via message multicasting.

Due to the recently rapid development of multimedia applications, multicast has become the critical technique in many network applications. In multicasting routing, the main objective is to send data from one or more sources to multiple destinations to minimize the usage of resources such as bandwidth, communication time, and connection costs. Chapter 70 discusses contemporary research concerning multicast congestion problems in different type of networks.

Recent progress in audio, video, and data storage technologies has given rise to a host of high-bandwidth real-time applications such as video conferencing. These applications require Quality of Service (QoS) guarantees from the underlying networks. Thus, multicast routing algorithms, which manage network resources efficiently and satisfy the QoS requirements, have come under increased scrutiny in recent years. Chapter 71 considers the problem of finding an optimal multicast tree with certain special characteristics. This problem is a generalization of the classical Steiner tree problem.

Scalability is especially critical for peer-to-peer systems. The basic idea of peer-to-peer systems is to have an open self-organizing system of peers that does not rely on any central server and where peers can join and leave, at will. This has the benefit that individuals can cooperate without fees or an investment in additional high-performance hardware. Also, peer-to-peer systems can make use of the tremendous amount of resources (such as computation and storage) that otherwise sit idle on individual computers when they are not in use by their owners. Chapter 72 seeks ways of implementing join, leave, and route operations so that for any sequence of join, leave, and route requests can be executed quickly; the degree, diameter, and stretch factor of the resulting network are as small as possible; and the *expansion* of the resulting network is as large as possible. Good approximate solutions to this multiobjective optimization problem are discussed in Chapter 72.

Scheduling problems modeling the broadcasting of data items over wireless channels are discussed in Chapter 73. The chapter covers exact and heuristic solutions for variants of this problem.

Microarrays have been evolving rapidly, and are among the most novel and revolutionary new biotechnologies. They allow us to monitor the expression of thousands of genes at once. With a single experiment billions of individual hypotheses can be tested. Chapter 74 presents three illustrative examples in the analysis of microarray data sets.

Chapter 75 considers two problems from computational biology, namely, primer selection and planted motif search. The closest string and the closest substring problems are closely related to the planted motif search problem. Representative approximation algorithms for these problems are discussed.

There are interesting algorithmic issues that arise when length constraints are taken into account in the formulation of a variety of problems on string similarity, particularly in the problems related to local alignment. Chapter 76 discusses these types of problems which have their roots and most striking applications in computational biology. Chapter 77 discusses approximation algorithms for the selection of robust tag single nucleotide polymorphisms (SNPs). This is a problem in human genomics that arises in the current experimental environment. Chapter 78 considers a sphere packing problem. Recent interest on this problem was motivated by medical applications in radiosurgery. Radiosurgery is a minimally invasive surgical procedure that uses radiation to destroy tumors inside the human body.

VLSI has produced some of the largest combinatorial optimization problems ever considered. Placement is one of the most difficult of these problems. Placement problems with over 10 million variables and constraints are not unusual, and problem sizes continue to grow with Moore's law. Realistic objectives and constraints for placement incorporate complex models of signal timing, power consumption, wiring routability, manufacturability, noise, temperature, etc. Chapter 79 considers VLSI placement algorithms.

Due to delay scaling effects in deep-submicron technologies, interconnect planning and synthesis are becoming critical to meeting VLSI chip performance targets with reduced design turnaround time. In particular, the global routing phase of the design cycle is receiving renewed interest, as it must efficiently handle increasingly more complex constraints for increasingly larger designs. Chapter 80 presents an integrated approach for congestion and timing-driven global routing, buffer insertion, pin assignment, and buffer/wire sizing. This is a multiobjective optimization problem.

Chapters 81–83 discuss game theory problems related to the Internet and scheduling. They deal with ways of achieving equilibrium. Issues related to algorithmic game theory, approximate economic equilibrium and algorithm mechanism design are discussed.

Over the last decade, the size of data seen by a computational problem has grown immensely. There appears to be more web pages than human beings, and web pages have been successfully indexed. Routers generate huge traffic logs, in the order of terabytes, in a short time. The same explosion of data is felt in observational sciences because our capabilities of measurement have grown significantly. Chapter 84 considers a processing mode where input items are not explicitly stored and the algorithm just passes over the data once.

A virtual community can be defined as a group of people sharing a common interest or goal who interact over a virtual medium, most commonly the Internet. Virtual communities are characterized by an absence of face-to-face interaction between participants which makes the task of measuring the trustworthiness of other participants harder than in nonvirtual communities. This is because of the anonymity that the Internet provides, coupled with the loss of audiovisual cues that help in the establishment of trust. As a result, digital reputation management systems are an invaluable tool for measuring trust in virtual communities. Chapter 85 discusses various systems which can be used to generate a good solution to this problem.

Chapter 86 considers the problem of approximating "colors." Several algorithmic methodologies are presented and evaluated experimentally. These algorithms include dimension weighted clustering approximation algorithms.

References

[1] Graham, R. L., Bounds for certain multiprocessing anomalies, *Bell Syst. Tech. J.*, 45, 1563, 1966.

[2] Graham, R. L., Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.*, 17, 263, 1969.

[3] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman, San Francisco, 1979.

[4] Coffman, E. G., Jr., Garey, M. R., and Johnson, D. S., An application of bin-packing to multiprocessor scheduling, *SIAM J. Comput.*, 7, 1, 1978.

[5] Friesen, D. K., Tighter bounds for the multifit processor scheduling algorithm, *SIAM J. Comput.*, 13, 170, 1984.

[6] Friesen, D. K. and Langston, M. A., Bounds for multifit scheduling on uniform processors, *SIAM J. Comput.*, 12, 60, 1983.

[7] Bar-Yehuda, R. and Even, S., A linear time approximation algorithm for the weighted vertex cover problem, *J. Algorithms*, 2, 198, 1981.

[8] Gonzalez, T. F., A simple LP-free approximation algorithm for the minimum web pages vertex cover problem, *Inform. Proc. Lett.*, 54(3), 129, 1995.

[9] Bar-Yehuda, R. and Even, S., A local-ratio theorem for approximating the weighted set cover problem, *Ann. Disc. Math.*, 25, 27, 1985.

[10] Bar-Yehuda, R. and Bendel, K., Local ratio: a unified framework for approximation algorithms, *ACM Comput. Surv.*, 36(4), 422, 2004.

[11] Schrijver, A., *Theory of Linear and Integer Programming*, Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley, New York, 2000.

[12] Vanderbei, R. J., *Linear Programming Foundations and Extensions,* International Series in Operations Research & Management Science, Vol. 37, Springer, Berlin, 2001.

[13] Sahni, S. and Gonzalez, T., P-complete approximation problems, *JACM*, 23, 555, 1976.

3

Restriction Methods

Teofilo F. Gonzalez
University of California, Santa Barbara

3.1 Introduction

Restriction is one of the most basic techniques to design approximation algorithms. The idea is to generate a solution to a given problem P by providing an optimal or suboptimal solution to a subproblem of P. By a subproblem of a problem P we mean restricting the solution space for P by disallowing a subset of the feasible solutions. The most common approach is to solve one subproblem, but there are algorithms that first solve several subproblems and the algorithm outputs the best of these solutions. An optimal or suboptimal solution to the subproblem(s) is generated by any of the standard methodologies.

This approach is in a sense the opposite of "relaxing a problem," i.e., augmenting the feasible solution space by including previously infeasible solutions. In this case one needs to solve a superproblem of P. An approximation algorithm for P solves the superproblem (optimally or suboptimally) and then transforms such solution to one that is feasible for P. Approximation algorithms based on the linear programming methodology fall under this category. There are many different conversion techniques including rounding, randomized rounding, etc. Chapters 4, 6, 7, and 12 discuss this approach in detail. Approximation algorithms based on both restriction and relaxation exist. These algorithms first restrict the solution space and then relaxes it. The resulting solution space is different from the original one.

In this chapter we discuss several approximation algorithms based on restriction. When designing algorithms of this type the question that arises is which of the many subproblems should be selected to provide an approximation for a given problem? One would like to select a subproblem that "works best." But what do we mean by a subproblem that works best? The one that works best could be a subproblem, which results in an approximation algorithm with smallest possible approximation ratio, or it could be a subproblem whose solution can be computed the fastest, or one may use some other criteria, for example, any of the ones discussed in Chapter 1. Perhaps "works best" should be with respect to a combination of different criteria. But even when using the approximation ratio as the only evaluation criteria for an algorithm, it is not at all clear how to select a subproblem that can be solved quickly and from which a best possible solution could be generated. These are the two most important properties when choosing a subproblem. By studying several algorithms based on restriction one learns why it works for these cases and then it becomes easier to find ways to approximate other problems.

The problems that we will discuss in this chapter to illustrate "restriction" are Steiner trees, the traveling salesperson, covering points by squares, rectangular partitions, and routing multiterminal nets. The Steiner tree and traveling salesperson problems (TSPs) are classical problems in combinatorial optimization. The algorithms that we discuss for the TSPs are among the best known approximation algorithms for any problem.

A closely related approach to restriction is *transformation-restriction.* The idea is to transform the problem instance to a restricted instance of the same problem. The difference is that the restricted problem instance is not a subproblem of original problem instance as in the case of restriction, but it is a "simpler" problem of the same type. In Section 3.5 we present algorithms based on this approach for routing multiterminal nets and embedding hyperedges in a cycle. The fully polynomial-time approximation scheme for the knapsack problem, based on rounding discussed in Chapter 10, is based on transformation-restriction. In Section 3.8 we summarize the chapter, and briefly discuss other algorithms based on restriction for path problems arising in computational geometry.

3.2 Steiner Trees

The Steiner tree problem is a classical problem in combinatorial optimization. Let us define the Steiner tree problem over an edge-weighted complete metric graph $G = (V, E, w)$, where V is the set of n vertices, E the set of $m = \frac{n^2-n}{2}$ edges, and $w : E \rightarrow R^+$ the weight function for the edges. Since the graph is metric the set of weights satisfies the triangle inequality, i.e., for every pair of vertices i, j, $w(i, j)$ is less than or equal to the sum of the weight of the edges in any path from vertex i to vertex j. The Steiner tree problem consists of a metric graph $G = (V, E, W)$ and a subset of vertices $T \subseteq V$. The problem is to find a tree that includes all the vertices in T plus some other vertices in the graph such that the sum of the weight of the edges in the tree is least possible. The Steiner tree problem in an NP-hard problem.

When $T = V$ the problem is called the minimum-weight (cost) spanning tree problem. By the 1960s there were several well-known polynomial-time algorithms to construct a minimum-weight spanning tree for edge-weighted graphs [1]. These simple greedy algorithms have low-order polynomial-time complexity bounds.

Given an instance of the metric graph Steiner tree problem ($G = (V, E, W)$, T) one may construct a minimum-weight spanning tree for the subgraph $G' = (T, E', W')$, where E' and W' include only the edges joining vertices in T. Clearly, this minimum-weight spanning tree is a restricted version of the Steiner tree problem and it seems a natural way to approximate the Steiner tree problem. This approach was analyzed in 1968 by E. F. Moore (see Ref. [2]) for the Steiner tree problem defined in metric space. The metric graph problem, we just defined, includes only a subset of all the possible points in metric space. E. F. Moore presented an elegant proof of the fact that in metric space (and also for metric graphs) $L_M < L_T \leq 2L_S$, where L_M, L_T, and L_S are the weight of a minimum-weight spanning tree, a minimum-weight tour (solution) for the TSP and minimum-weight Steiner tree for any set of points P, respectively. We will define the TSP in the next section. Since every spanning tree is a Steiner tree, the above bounds show that when using a minimum-weight spanning tree to approximate the Steiner tree results in a solution whose weight is at most twice the weight of an optimal Steiner tree. In other words, any algorithm that generates a minimum-weight spanning tree is a 2-approximation algorithm for the Steiner tree problem. Furthermore, this approximation algorithm takes the same time as an algorithm that constructs a minimum-weight spanning trees for edge-weighted graphs [1], since such an algorithm can be used to construct an optimal spanning tree for a set of points in metric space. The above bound is established by defining a transformation from any minimum-weight Steiner tree into a TSP tour in such a way that $L_T \leq 2L_S$ [2]. Then by observing that the deletion of an edge in an optimum tour to the TSP results in a spanning tree, one has $L_M < L_T$. The proof is identical to the one given in the next section where we show this result, but starting from a minimum-weight spanning tree.

3.3 Traveling Salesperson Tours

The TSP has been studied for several decades [3]. There are many variations of this problem. One of the simplest versions of the problem consists of an edge-weighted complete graph and the problem is to find a minimum-weight tour that starts and ends at vertex one and visits every vertex *exactly* once. The weight of a tour is the sum of the weight of the edges in the tour. Sahni and Gonzalez [4] (see Chapter 1) show that the constant-ratio approximation problem is NP-hard, i.e., if for any constant c there is a polynomial-time algorithm with approximation ratio c then $P = NP$. In this section we discuss approximation algorithms for the TSP defined over complete metric graphs. These algorithms are among the best known approximation algorithms for any problem. The "double-minimum-weight spanning tree" (DMWST) approximation algorithm that we discuss in this section is widely known, and it is based on the constructive proof for the approximation algorithm discussed in the previous section developed for the Steiner tree problem by E. F. Moore. Additional constant-ratio approximation algorithms for this version of the TSP were developed by Rosenkrantz et al. [5]. These algorithms as well the DMWST algorithm have an approximation ratio of $2 - 1/n$ and take $O(n^2)$ time. Since the graph is complete, the time complexity is linear with respect to the number of edges in the graph. After presenting this result we discuss the improved approximation algorithm by Christofides [6]. This algorithm has a smaller approximation ratio, but its time complexity grows faster than that of the previous algorithms.

In the literature you will find that the TSP is also defined with tours visiting each vertex *at least* once. We now show that both versions of the TSP defined over metric graphs are equivalent problems. Consider any optimal tour R where some vertices are visited more than once. Let vertex i be a vertex visited more than once. Let vertices j and k be visited just before and just after vertex i. Delete from the tour the edges $\{j, i\}$ and $\{i, k\}$ and add edge $\{j, k\}$. Because the graph is metric the tour weight will stay the same or decrease. If it decreases, then it contradicts the optimality of R. So the weight of the tour must be the same as before. After applying this transformation until it is no longer possible we obtain a tour R' in which every vertex is visited exactly once and the weight of R' is identical to that of R. Since every tour that visits every vertex exactly once also visits every vertex at least once, it follows that both versions of the problem for metric graphs have the same optimal tour weight, i.e., both problems are equivalent. Since for the TSP defined over metric graphs both versions of the problem are equivalent, for convenience we use the definition of tours to visit each vertex at least once.

Now suppose that you have an optimal tour S for an instance I of the TSP. Applying the above transformation we obtain an optimal tour S' in which every vertex is visited exactly once. Deleting an edge from the tour results in a spanning tree. Therefore, the weight of a minimum-weight spanning tree is a lower bound for the weight of an optimal tour. The questions are: How good of a lower bound is it? How can one construct a tour from a spanning tree?

How can we find a tour from a spanning tree T? Just draw the spanning tree in the plane with a vertex as its root and construct a tour by visiting each edge in the tree T twice as illustrated in Figure 3.1. A more

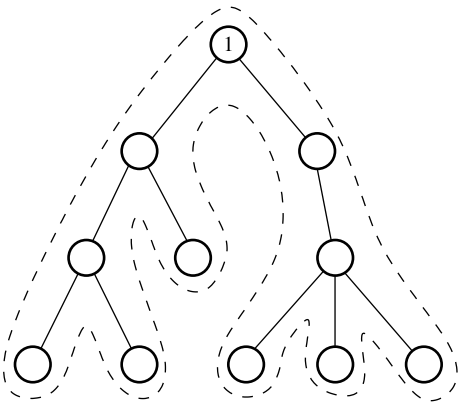


FIGURE 3.1 Spanning tree (solid lines) and tour constructed (broken lines).

formal approach is to construct an *Euler circuit* in the multigraph (graph with multiple edges between vertices) consisting of two copies of the edges in T. An Euler tour (or circuit) is a path that starts and ends at the same vertex and visits every edge in the multigraph once. An Euler tour always exists for the multigraphs we have defined because these multigraphs are connected and all their nodes are of even degree (the number of edges incident to each vertex is even). These multigraphs are called *Eulerian*, and an Euler tour can be constructed in linear time with respect to the number of nodes and edges in the multigraph [7].

The approximation algorithm, which we refer to as *DMWST*, constructs a minimum weight spanning tree, makes a copy of all the edges in the tree, and then generates a tour from this tree with weight equal to twice the weight of a minimum weight spanning tree. We established before that an optimal tour has weight greater than the weight of a minimum weight spanning tree, it then follows that the weight of the tour that the DMWST algorithm generates is at most twice the weight of an optimal tour for G. Therefore, algorithm DMWST generates 2-approximate solution. Actually the ratio is $2-1/n$, which can be established when the edge deleted for an optimal tour to obtain a spanning tree is one with largest weight. The time complexity of the algorithm is bounded by the time complexity for generating a minimum weight spanning tree, since an Euler tour can be constructed in linear time with respect to the number of edges in the spanning tree. We formalize these results in the following theorem.

Theorem 3.1

For the metric traveling salesperson problem, algorithm DMWST generates a tour with weight at most $(2-1/n)$ *times the weight of an optimal tour. The time complexity of the algorithm is* $O(n^2)$ *time, which is linear time with respect to the number of edges in the graph.*

Proof

The proof for the approximation ratio follows from the above discussion. As Fredman and Tarjan [8] point out, implementing Prim's minimum weight spanning tree algorithm by using Fibonacci heaps results in a minimum weight spanning tree algorithm that takes $O(n \log\, n + m)$ time. Since the graph is complete, the time complexity is $O(n^2)$, which is linear with respect to the number of edges in the graph. □

So what is the restriction in the above algorithms? We are actually *restricting* tours for the TSP to traverse the least possible number of *different* edges, though a tour may traverse some of these edges more than once. The minimum number of different edges in G is $n-1$ and they form a spanning tree. It is therefore advantageous to select the edges in a spanning tree of least possible total weight. This justifies the use of a minimum-weight spanning tree. This is another way to think about the design of the DMWST algorithm.

Christofides [6] modified the above approach so that the tours generated have total weight within 1.5 times the weight of an optimal tour. However, the currently fastest implementation of this procedure takes $O(n^3)$ time. His modification is very simple. First observe that there are many different ways to transform a spanning tree into an Eulerian multigraph. All possible augmentations must include at least one edge incident to every odd degree vertex in the spanning tree. Let N be the set of odd degree vertices in the spanning tree. Christofides, idea is to transform the spanning tree into an Eulerian multigraph by adding the least number of edges with the least possible total weight. He showed that such set of edges is a minimum weight complete matching on the graph G_N induced by the set of vertices N in G. A *matching* is a subset of the edges in a multigraph, no two of which are incident upon the same vertex. A matching is *complete* if every node has an edge in the matching incident to it, and the weight of a matching is the sum of the weights of the edges in it. A minimum weight complete matching can be constructed in polynomial time. The edges in the complete matching plus the ones in the spanning tree form an Eulerian multigraph, and Christofides' algorithm generates as its solution an Euler tour of this multigraph.

To establish the 1.5 approximation bound we observe that an optimal tour can be transformed without increasing its total weight into another tour that visits only the vertices in N because the graph is metric. One can partition the edges in this restricted tour into two sets such that each set is a complete matching for the restricted graph. One set contains the even-numbered edges in the tour and the other set the odd-numbered edges. Since a minimum weight complete matching for G_N has total weight smaller than the above two matchings, it then follows that the minimum weight complete matching has total weight at most half of the weight of an optimal tour. Therefore, the edges in the tour constructed by Christofides' algorithm have weight at most 1.5 times the weight of an optimal tour. The time complexity for Christofides' algorithm is $O(n^3)$ and it is dominated by the time required to construct a minimum weight complete matching [9,10]. We formalize this result in the following theorem whose proof follows from the above discussion.

Theorem 3.2 [6]

For the metric traveling salesperson problem, Christofides' algorithm generates a tour with weight at most 1.5 *times the weight of an optimal tour. The time complexity of the algorithm is* $O(n^3)$.

This approach is similar to the one employed by Edmonds and Johnson [11] for the *Chinese Postman Problem.* Given an edge-weighted connected undirected graph, the Chinese Postman problem is to construct a minimum-weight cycle, possibly with repeated edges, which contains every edge in the graph. The currently best algorithm to solve this problem takes $O(n^3)$ time, and it uses shortest paths and weighted matching algorithms. There are asymptotically faster algorithms when the graphs are sparse and weight of the edges are integers.

3.4 Covering Points by Squares

Given a set of n points, $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, in two-dimensional space and an integer D, the CS_2 problem is to find the least number of $D \times D$ squares to cover P. The CS_2 problem as well as the problem of covering by disks have been shown to be NP-hard [12]. Approximation algorithm for these problems as well as their generalizations to multidimensional space have been developed [13,14]. All of these problems find applications in several research areas [12,15,16]. The most popular application is to find the least number of emergency facilities such that every potential patient lives at a distance at most D from at most one facility. This application corresponds to covering by the least number of disks with radius D.

We discuss in this section a simple approximation algorithm based on restriction for the CS_2 problem. Assume without loss of generality that $x_i \geq 0$ and $y_i \geq 0$ and that at least one of the points has x-coordinate value of zero. Define the function $I_x(P_i) = \lfloor x_i/D \rfloor$. For $k \geq 0$, *band k* consists of all the points with $I_x(P_i) = k$.

The restriction to the solution space is to only allow feasible solutions where each square covers points from only one band. Note that an optimal solution to the CS_2 problem does not necessarily satisfy this property. For example, the instance with $P_1 = (0.1, 1.0)$, $P_2 = (0.1, 2.0)$, $P_3 = (1.1, 0.9)$, $P_4 = (1.1, 2.1)$, and $D = 1$ has two squares in optimal cover. The first square covers points P_1 and P_3, and the second covers P_2 and P_4. However an optimal cover for the points in band 0 (i.e., P_1 and P_2) is one square and the one for the points in band 1 (i.e., P_3 and P_4) is two squares. So an optimal cover to the restricted problem has three squares, but an optimal cover for the CS_2 problem has two squares.

One reason for restricting the solution space in this way is that an optimal cover for any given band can be easily generated by a greedy procedure in $O(n \log n)$ time [14]. A greedy approach places a square as high as possible provided it includes the bottommost point in the band as well as all other points in the band at a vertical distance at most 1 from a bottommost point. All the points covered by this square are removed and the procedure is repeated until all the points have been covered. One can easily show that this is an optimal cover by transforming any optimal solution for the band, without increasing the number of squares, to the cover generated by the greedy algorithm. By using elaborate data structures, Gonzalez [14] showed that the greedy algorithm can be implemented to take $(n \log s)$, where s is the number of squares in an optimal solution. Actually a method that uses considerable more space can be used to solve the problem in $O(n)$ time [14].

The solution generated by our algorithm for the whole problem is the union of the covers for each of the bands generated by the greedy method. Let $\hat{f} = E + O$ be the total number of squares, where $E(O)$ is the number of squares for the even (odd-)-numbered bands. We claim that an optimal solution to the CS_2 problem has at least $max\{E, O\}$ squares. This follows from the fact that an optimal solution for the even (odd-)-numbered bands is $E(O)$ because it is not possible for a square to cover points from two different even (odd-)-numbered bands. Therefore, $\frac{\hat{f}_I}{f_I^*} \leq 2$, where f_I^* is the number of squares in an optima solution for problem instance I. This result is formalized in the following theorem whose proof follows from the above discussion.

Theorem 3.3

For the CS_2 problem the above procedure generates a cover such that $\frac{\hat{f}_I}{f_I^*} \leq 2$ in $O(n \log s)$ time, where s is the number of squares in an optimal solution.

A polynomial-time approximation scheme for the generalization of the CS_2 to d dimensions (the CS_d problem) is discussed in Chapter 9. The idea is to generate a set of solutions by shifting the bands by different amounts and then selecting as the solution the best cover computed by the algorithm. This approach is called *shifting* and was introduced by Hochbaum and Maass [13].

3.5 Rectangular Partitions

The minimum edge-length rectangular partition, RG_P problem has applications in the area of computer-aided design of integrated circuits and systems. Given a rectangle R with interior points P, the RG_P problem is to introduce a set of interior lines segments with least total length such that every point in P is in at least one of the partitioning line segments, and R is partitioned into rectangles. Figure 3.2(a) shows a problem instance I and Figure 3.2(b) shows an optimal rectangular partition for the problem instance I.

A rectangular partition E is said to have a *guillotine cut* if one of the vertical or horizontal line segments partitions the rectangle into two rectangles. A rectangular partition E is said to be a *guillotine partition* if either E is empty, or E has a guillotine cut and each of the two resulting rectangular partitions is a guillotine partition.

Finding an optimal rectangular partition is an NP-hard problem [17]. However, an optimal guillotine partition can be constructed in polynomial time. Therefore, it is natural to restrict the solution space to guillotine partitions when approximating rectangular partitions.

In Chapter 54 we prove that an optimal guillotine partition has total edge length, which is at most twice the length of an optimal rectangular partition. Gonzalez and Zheng [18] presented a complex proof that shows that bound is just 1.75. In Chapter 54 we also explain the basic ideas behind the proof of the approximation ratio of 1.75. This approach has been extended to the multidimensional version of this problem by Gonzalez et al. [19].

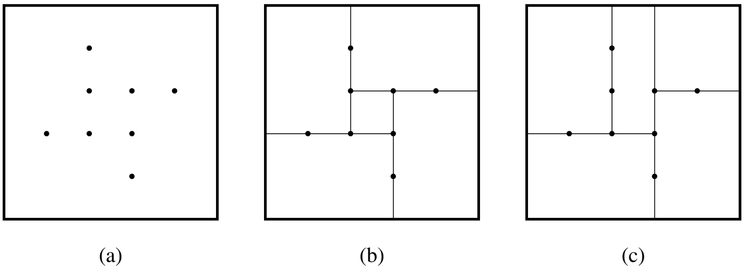


FIGURE 3.2 (a) Instance I of the RG_P problem. (b) Rectangular partition for the instance I. (c) Guillotine partition for the instance I.

An optimal guillotine partition can be constructed in $O(n^5)$ time via dynamic programming. When n is large this approach is not practical. Gonzalez et al. [20] showed that suboptimal guillotine partitions that can be constructed in $O(n \log n)$ time generate solutions with total edge length at most four times the length of an optimal rectangular partition. As in the case of optimal guillotine partitions this result has been extended to the multidimensional version of the problem [20]. Clearly, neither of the methods dominates the other when considering both the approximation ratio and the time complexity bound.

Chapter 42 discusses how more general guillotine cuts can be used to develop a polynomial time approximation scheme (PTAS) for the TSP in two-dimensional space. Chapter 51 discusses this approach for the TSP and Steiner tree problems.

3.6 Routing Multiterminal Nets

Let R be a rectangle whose sides lie on the two-dimensional integer grid. A subset of grid points on the boundary of R that do not include the corners of R is denoted by S and its grid points are called *terminal points*. Let n be the number of terminal points, i.e., the cardinality of set S, and let $N_1, N_2, \ldots, N_m$ a partition of S such that each set N_i includes at least two terminal points. Each set N_i is called a *net* and the problem is to make all the terminal points electrically common by introducing a set wire segments. Terminal points from different nets should not be made electrically common. The wire segment must be along the grid lines outside R with at most one wire segment assigned to each grid edge. When the grid edges incident to a grid point belong to wire segments from two nets, the two wires must cross. In other words, dog-legs (wires from two nets bending at a grid point) are not allowed. The main reasons are that dog-legs would complicate the layer assignment without improving the layout area.

There are two layers available for the wires. Since dog-legs are not allowed, the layer assignment for the wire segments is straightforward. All horizontal wire segments are assigned to one layer and all the vertical ones are assigned to the other. A vertical and horizontal wire segment with a common grid point can be made electrically common by introducing a via for the connection of the wires at that grid point.

The Multiterminal net routing Around a Rectangle (*MAR*) problem is given a rectangle R and a set of nets, find a layout, subject to the constraints defined above, that fits inside a rectangle with least possible area. Constructing a layout in this case reduces to just finding the wire segments for each net along the grid lines (without dog-legs) outside R, since the layer assignment is straightforward.

Developing a constant-ratio approximation algorithm for this problem is complex because the objective function depends on the product of two values, rather than just one value as in most other problems. Gonzalez and Lee [21] developed a linear-time algorithms for the *MAR* problem when every net consists of two terminal points. It is conjectured that the problem is NP-hard when the nets have three terminal points each. Gonzalez and Lee [21,22] developed constant-ratio approximation algorithms for the *MAR* problem [22,23]. The approximation ratios for these algorithms are 1.69 [22] and 1.6 [23]. The approach is to partition the set of nets into groups and then route each group of nets independently of each other. Some of the groups are routed optimally. Since the analysis of the approximation ratio for these algorithms is complex, in this section we only analyze the case when the nets contain one terminal point on the top side of R and one or more terminal points on the bottom side of R. The set of these nets is called N_{TB}. The algorithm to route the N_{TB} nets is based on restriction and it is quite interesting. Readers interested in additional details are referred to Refs. [22,23].

Let n_{TB} be the number of N_{TB} nets. Let E be an optimal area layout for all the nets and let D be E except that the set of nets in N_{TB} are all connected by a path that crosses the left side of R. In this case the layout for the nets N_{TB} is restricted (only paths that cross the left side of R are allowed). We use $H_E(TB)$ ($H_D(TB)$) to denote the height of the layout $E(D)$ on the top plus the corresponding height on the bottom side of R. To simplify the analysis, let us assume that every net in N_{TB} is connected in E by a path that either crosses the left or right (but not both) sides of R. Gonzalez and Lee [23] explain how to modify the analysis when some of these nets are connected by paths that cross both the left and right sides of R.

By *reversing* the connecting path for a net in N_{TB} we mean to connect the net by a path that crosses the opposite side of R, i.e., if it crossed the left side of R it will now cross the right side, or vice versa. When we reverse the connecting path for a net the height on the top side plus the bottom side of R increases by at most two. We say that connecting paths for two N_{TB} nets *cross on the top side* of R when their contribution to the height of the assignment is two for at least one point in between two terminal points. When we *interchange* the connecting paths for two N_{TB} nets that cross on the top side of R we mean reversing both connecting paths. An interchange increases by at most two the height on the top side plus the bottom side of R.

We transform E to D by reversals to quantify the difference in heights between E and D. The largest increase in height is when all the N_{TB} nets are connected in E by paths that cross the right side of R. In this case we need to reverse all the connecting paths for the N_{TB} nets, so $H_D(TB) \leq H_E(TB) + 2n_{TB}$. When one plugs this in the analysis for the whole problem it results in an algorithm with an approximation ratio greater than 2.

A better approach is to use the following restriction. All the connecting paths for the N_{TB} nets are identical, and either they cross the left or the right side of R. In this case we construct two different layouts. Let $D_l(D_r)$ be E except that all the nets in N_{TB} are connected by a path crossing the left (right) side of R. Let M be a minimum area layout between D_l and D_r. In E let $l(r)$ be the number of N_{TB} nets connected by a path crossing the left (right) side of R. By reversing the minimum of $\{l, r\}$ paths it is possible to transform E to D_l or D_r. Therefore, $H_M(TB) \leq H_E(TB) + n_{TB}$, which is better by 50% than for the assignment D defined above.

It is obvious that by trying more alternatives one can obtain better solutions. Let us partition the set of nets N_{TB} into two groups, S_l and S_r. The set S_l contains the $\frac{n_{TB}}{2}$ nets in N_{TB} whose terminal point on the top side of R is closest to the left side of R, and set S_r contains the remaining ones. For $i, j \in \{l, r\}$ let D_{ij} be E except that all the nets in S_l are connected by paths that cross the "i" side of R and all the nets in S_r are connected by paths that cross the "j" side of R. Let P be a minimum area layout among D_{ll}, D_{lr}, D_{rl}, and D_{rr}. Let $l_1(r_1)$ be the number of nets in S_l connected by a path that crosses the left side of R. We define l_2 and r_2 similarly, but using set S_l. We show in the following lemma that $H_P(TB) \leq H_E(TB) + \frac{3}{4}n_{TB}$.

Lemma 3.1

Let P and E be the assignments defined above. Then $H_P(TB) \leq H_E(TB) + \frac{3}{4}n_{TB}$.

Proof

The proof is by contradiction. Suppose that $H_P(TB) > H_E(TB) + \frac{3}{4}n_{TB}$. There are two cases depending on the values of r_1 and l_2.

Case 1: $r_1 \geq l_2$. To transform assignment E to D_{lr} we need to interchange l_2 connecting paths that cross on the top side of R and reverse $r_1 - l_2$ connecting paths. Therefore, $H_{D_{lr}}(TB) \leq H_E(TB) + 2r_1$. Since $H_{D_{lr}}(TB) \geq H_P(TB) > H_E(TB) + \frac{3}{4}n_{TB}$, we know that $2r_1 > \frac{3}{4}n_{TB}$, which is equivalent to $r_1 > \frac{3}{8}n_{TB}$. Since $r_1 + l_1 = \frac{1}{2}n_{TB}$, we know that $l_1 < \frac{1}{8}n_{TB}$.

To transform assignment E to D_{rr} we need to reverse $l_1 + l_2$ connecting paths. Therefore, $H_{D_{rr}}(TB) \leq H_E(TB) + 2l_1 + 2l_2$. Since $H_{D_{rr}}(TB) \geq H_P(TB) > H_E(TB) + \frac{3}{4}n_{TB}$, we know that $l_1 + l_2 > \frac{3}{8}n_{TB}$. Applying the same argument to assignment D_{rl}, we know $l_1 + r_2 > \frac{3}{8}n_{TB}$. Adding these two last inequalities and substituting the fact that $l_2 + r_2 = \frac{1}{2}n_{TB}$, we know that $l_1 > \frac{1}{8}n_{TB}$. This contradicts our previous finding that $l_1 < \frac{1}{8}n_{TB}$.

Case 2: $r_1 < l_2$. A contradiction in this case can be obtained applying similar arguments.

It must then be that $H_P(TB) \leq H_E(TB) + \frac{3}{4}n_{TB}$. □

For three groups, rather than two, Gonzalez and Lee [22] showed that $H_P(TB) \leq H_E(TB) + \frac{2}{3}n_{TB}$, where P is the best of the eight assignments generated. This is enough to prove the approximation ratio of 1.69 for the *MAR* problem. If instead of three groups one uses six, one can prove $H_P(TB) \leq H_E(TB) + 0.6n_{TB}$, where P is the best of the 64 assignments generated. In this case, the approximation ratio for the *MAR* problem is 1.6. Interestingly, partitioning into more groups results in smaller bounds for this group, but does not reduce the approximation ratio for the *MAR* problem because the routing of other nets becomes the bottleneck. We state Gonzalez and Lee's theorem without a proof. Readers interested in the proof are referred to Ref. [23].

Theorem 3.4

For the MAR problem the procedure given in Ref. [23] generates a layout with area at most 1.6 *times the area of an optimal layout in* $O(nm)$ *time.*

An interesting observation is that the proof that the bound $H_P(TB) \leq H_E(TB) + (1.6)n_{TB}$ holds can be carried out automatically by solving a set of linear programming problems. The linear programming problems find the ratios for l_i and r_i such that the minimum increase from E to one of the layouts is maximized. Note that some of the "natural" constraints of the problem are in terms $\max\{r_1, l_2\}$, which makes the solution space nonconvex. However by replacing it with inequalities of the form $r_1 \leq l_2$ and $r_1 > l_2$ we partition the optimization region into several convex regions. By solving a set of linear programming problems (one for each convex region) the maximum possible increase can be computed.

3.7 Variations on Restriction

A closely related approach to restriction is to generate a solution by solving a restricted problem instance constructed from the original instance. We call this approach *transformation-restriction*. For example, consider the routing multiterminal nets around a rectangle discussed in Section 3.6. Remember that there are n terminal points and m nets. Suppose that we break every net i with k_i points into k_i nets with two terminal points each. The k nets consist of adjacent terminal points of the net. In order for these k_i nets to have different terminal points we make a copy of each terminal point at half-integer points next to the old ones. Note that a new grid needs to be redefined to include the half-integer points without introducing more horizontal (vertical) routing tracks above or below (to the left or right) of R. Figure 3.3(b) shows the details. The resulting 2-terminal net problems can be solved in linear time using the optimal algorithm developed by Gonzalez and Lee [21]. A solution to this problem can be easily transformed into a solution to the original problem after deleting the added terminal points as well as some superfluous connections. This algorithm generates a layout whose total area is at most 4 times the area of an optimal layout. Furthermore, the layout can be constructed in $O(n)$ time. With respect to the approximation ratio Gonzalez and Lee's algorithms [22,23] are better, but these algorithms take $O(nm)$ time, whereas the simple algorithm in this section takes linear time.

3.7.1 Embedding Hyperedges in a Cycle

In this subsection we present an approximation algorithm for Embedding Hyperedges in a Cycle so as to Minimize the Congestion (EHCMC). As pointed out in Chapter 70, this problem has applications in the area of design automation and parallel computing. As input we are given a hypergraph $G = (V, H)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set vertices and $H = \{h_1, h_2, \ldots, h_m\}$ the set of hyperedges (or subsets with at least two elements of the set V). Traversing the vertices $v_1, v_2, \ldots, v_n$ in the clockwise direction

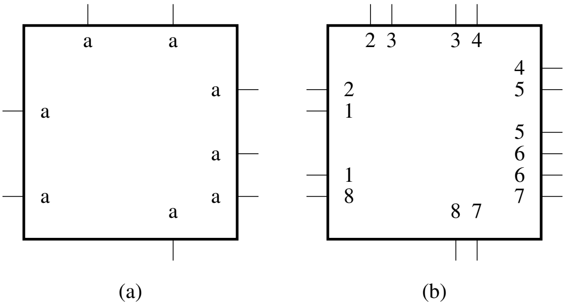


FIGURE 3.3 (a) Net with k-terminal points. (b) Resulting k 2-terminal nets.

forms a cycle, which we call C. Let v_t and v_s be two vertices in h_i such that v_s is the next vertex in h_i in clockwise direction from v_t. Then the pair (v_s, v_t) for hyperedge h_i defines the connecting path for h_i that begins at vertex v_s then proceeds in the clockwise direction along the cycle until reaching vertex v_t. Every edge e in the cycle that is visited by the connecting path formed by pair (v_s, v_t) is said to be *covered* by the connecting path. The EHCMC problem consists of finding a connecting path c_i for every hyperedge h_i such that the maximum congestion of an edge in C is least possible, where the congestion of an edge e in cycle C is the number of connecting paths that include edge e.

Ganley and Cohoon [24] showed that when the maximum congestion is bounded by a fixed constant k, the EHCMC problem is solvable in polynomial time. But, the problem is NP-hard when there is no constant bound for k. Frank et al. [25] showed that when the hypergraph is a graph the EHCMC problem can be solved in polynomial time. We call this problem the Embedding Edges in a Cycle to Minimize Congestion (EECMC). In this section we present the simple linear-time algorithm with an approximation ratio of 2 for the EHCMC problem developed by Gonzalez [26].

The algorithm based on *transformation-restriction* for this problem is simple and uses the same approach as in the previous subsection. This general approach also works for other routing problems. A hyperedge with k vertices $x_1, x_2, \ldots, x_k$, appearing in that order around the cycle C is decomposed into the following k edges $\{x_1, x_2\}, \{x_2, x_3\}, \ldots, \{x_{k-1}, x_k\}, \{x_k, x_1\}$. Note that in this case we do not need to introduce additional vertices as in the previous subsection because a vertex may be part of several hyperedges. The decomposition transforms the problem into an instance of the EECMC problem, which can be solved by the algorithm given in Ref. [25]. From this embedding we can construct an embedding for the original problem instance after deleting some superfluous edges in the embedding. The resulting embedding can be easily shown to have congestion of at most twice the one in an optimal solution X. This is because there is a solution S to the EECMC problem instance in which every connecting path Y in X can be mapped to a set of connecting paths in S with the property that if the connecting path Y contributes one unit to the congestion of an edge e, then the set of connecting paths in S contributes 2 units to the congestion of edge e. Furthermore, each connecting path in S appears in one mapping. The time complexity of the algorithm is $O(n)$.

3.8 Concluding Remarks

We have seen several approximation algorithms based on restriction. As we have seen the restricted problem may be solved optimally or suboptimally as in Section 3.5. One generates solutions closer to optimal, whereas the other generates the solutions faster. These are many more algorithms based on this technique. For example, some computational geometry problems where the objective function is in terms of distance have been approximated via restriction [27–30]. These type of problems allow feasible solutions to be any set of points along a given set of line segments. A restricted problem allows only a set of points (called *artificial points*) to be part of a feasible solution. The more artificial points, the smaller the approximation ratio of the solution; however, it will take longer to solve the restricted problem.

There are problems for which it is not known whether or not there is a constant-ratio approximation algorithm. However, heuristics based on restriction are used to generate good solutions in practice. One such problems is discussed in Chapter 73.

A closely related approach to restriction is *transformation-restriction.* The idea is to transform the problem instance to a restricted instance of the same problem. The difference is that the restricted problem instance is not a subproblem of original problem instance as in the case of restriction. In this chapter we applied this approach to a couple of problems.

Approximations algorithms that are based on restriction and relaxation exist. These algorithms first restrict the solution space and then relaxes it resulting in a solution space that is different from the original one. Gonzalez and Gonzalez [31] have applied this approach successfully to the minimum edge length corridor problem.

References

[1] Sahni, S., *Data Structures, Algorithms, and Applications in C++*, 2nd ed., Silicon Press, Summit, NJ, 2005.

[2] Gilbert, E. N. and Pollak, H. O., Steiner minimal trees, *SIAM J. Appl. Math.*, 16(1), 1, 1968.

[3] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B., Eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, New York, 1985.

[4] Sahni, S. and Gonzalez, T., P-complete approximation problems, *JACM*, 23, 555, 1976.

[5] Rosenkrantz, R., Stearns, R., and Lewis, L., An analysis of several heuristics for the traveling salesman problem, *SIAM J. Comput.,* 6(3), 563, 1977.

[6] Christofides, N., Worst-case analysis of a new heuristic for the traveling salesman problem, Technical report 338, Grad School of Industrial Administration, CMU, 1976.

[7] Weiss, M. A., *Data Structures & Algorithms in C++,* 2nd ed., Addison-Wesley, Reading, MA, 1999.

[8] Fredman, M. and Tarjan, R. E., Fibonacci heaps and their uses in improved network optimization algorithms, *JACM,* 34(3), 596, 1987.

[9] Gabow, H. N., A scaling algorithm for weighted matching on general graphs, *Proc. of FOCS,* 1985, p. 90.

[10] Lawler, E. L., *Combinatorial Optimization: Networks and Matroids,* Holt, Rinehart and Winston, New York, 1976.

[11] Edmonds, J. and Johnson, E. L., Matching Euler tours and the Chinese postman, *Math. Program.,* 5, 88, 1973.

[12] Flower, R. J., Paterson, M. S., and Tanimoto, S. L., Optimal packing and covering in the plane are NP-complete, *Inf. Process. Lett.,* 12, 133, 1981.

[13] Hochbaum, D. S. and Maass, W., Approximation schemes for covering and packing problems in image processing and VLSI, *JACM,* 32(1), 130, 1985.

[14] Gonzalez, T. F., Covering a set of points in multidimensional space, *Inf. Process. Lett.,* 40, 181, 1991.

[15] Tanimoto, S. L., Covering and indexing an image subset, *Proc. IEEE Conf. on Pattern Recognition and Image Processing,* 1979, p. 239.

[16] Tanimoto, S. L. and Fowler, R. J., Covering image subsets with patches, *Proc. 5th Int. Conf. on Pattern Recognition,* 1980, p. 835.

[17] Lingas, A., Pinter, R. Y., Rivest, R. L., and Shamir, A., Minimum edge length partitioning of rectilinear polygons, *Proc. 20th Allerton Conf. on Communication, Control, and Computing,* Monticello, Illinois, 1982.

[18] Gonzalez, T. F. and Zheng, S. Q., Improved bounds for rectangular and guillotine partitions, *J. Symb. Comput.,* 7, 591, 1989.

[19] Gonzalez, T. F., Razzazi, M., Shing, M., and Zheng, S. Q., On optimal d-guillotine partitions approximating hyperrectangular partitions, *Comput. Geom.: Theory Appl.,* 4(1), 1, 1994.

[20] Gonzalez, T. F., Razzazi, M., and Zheng, S. Q., An efficient divide-and-conquer algorithm for partitioning into d-boxes, *Int. J. Comput. Geom. Appl.,* 3(4), 417, 1993.

[21] Gonzalez, T. F. and Lee, S. L., A linear time algorithm for optimal wiring around a rectangle, *JACM,* 35(4), 810, 1988.

[22] Gonzalez, T. F. and Lee, S. L., Routing multiterminal nets around a rectangle, *IEEE Trans. Comput.,* 35(6), 543, 1986.

[23] Gonzalez, T. F. and Lee, S. L., A 1.60 approximation algorithm for routing multiterminal nets around a rectangle, *SIAM J. Comput.,* 16(4), 669, 1987.

[24] Ganley, J. L. and Cohoon, J. P., Minimum-congestion hypergraph embedding in a cycle, *IEEE Trans. Comput.,* 46(5), 600, 1997.

[25] Frank, A., Nishizeki, T., Saito, N., Suzuki, H., and Tardos, E., Algorithms for routing around a rectangle, *Discrete Appl. Math.,* 40(3), 363, 1992.

[26] Gonzalez, T. F., Improved approximation algorithms for embedding hypergraphs in a cycle, *Inf. Process. Lett.*, 67, 267, 1998.

[27] Papadimitriou, C. H., An algorithm for shortest path motion in three dimensions. *Inf. Process. Lett.*, 20, 259, 1985.

[28] Choi, J., Sellen, J., and Yap, C. K., Approximate Euclidean shortest path in 3-space, *Int. J. Comput. Geom. Appl.*, 7(4), 271, 1997.

[29] Choi, J., Sellen, J., and Yap, C. K., Approximate Euclidean shortest path 3-space, *Proc. 10th Annual Symp. on Computational Geometry,* 1994, p. 41.

[30] Bhosle, A. M. and Gonzalez, T. F., Exact and approximation algorithms for finding an optimal bridge connecting two simple polygons, *Int. J. Comput. Geom. Appl.,* 15(6), 609, 2005.

[31] Gonzalez, A. and Gonzalez, T. F., Approximation algorithms for minimum edge-length corridors, Technical Report, CS Department, UCSB, 2007.

4

Greedy Methods

Samir Khuller
University of Maryland

Balaji Raghavachari
University of Texas at Dallas

Neal E. Young
University of California at Riverside

4.1 Introduction

Greedy algorithms can be used to solve many optimization problems exactly and efficiently. Examples include classical problems such as finding minimum spanning trees and scheduling unit length jobs with profits and deadlines. These problems are special cases of finding a maximum- or minimum-weight basis of a *matroid*. This well-studied problem can be solved exactly and efficiently by a simple greedy algorithm [1,2].

Greedy methods are also useful for designing efficient *approximation* algorithms for intractable (i.e., NP-hard) combinatorial problems. Such algorithms find solutions that may be suboptimal, but still satisfy some performance guarantee. For a minimization problem, an algorithm has *approximation ratio* α, if, for every instance I, the algorithm delivers a solution whose cost is at most $\alpha \times \text{OPT}(I)$, where $\text{OPT}(I)$ is the cost of an optimal solution for instance I. An α-approximation algorithm is a polynomial-time algorithm with an approximation ratio of α.

In this chapter, we survey several NP-hard problems that can be approximately solved via greedy algorithms. For a couple of fundamental problems, we sketch the proof of the approximation ratio. For most of the other problems that we survey, we give brief descriptions of the algorithms and citations to the articles where these results were reported.

4.2 Set Cover

We start with SET COVER, perhaps one of the most elementary of the NP-hard problems. The problem is defined as follows. The input is a set $X = \{x_1, x_2, \ldots, x_n\}$ of elements and a collection of sets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ whose union is X. Each set S_i has a weight of $w(S_i)$. A *set cover* is a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $\bigcup_{S_j \in \mathcal{S}'} S_j = X$. Our goal is to find a set cover $\mathcal{S}' \subseteq \mathcal{S}$ so as to minimize $w(\mathcal{S}') = \sum_{S_i \in \mathcal{S}'} w(S_i)$. In other words, we wish to choose a minimum-weight collection of subsets that covers all the elements.

Intuitively, for a given weight, one prefers to choose a set that covers most of the elements. This suggests the following algorithm: Start with an empty collection of sets, then repeatedly add sets to the collection, each time adding a set that minimizes the cost per newly covered element (i.e., the set that minimizes the weight of the set divided by the number of its elements that are not yet in any set in the collection).

4.2.1 Algorithm for Set Cover

Next we prove that this algorithm has approximation ratio $H(|S_{\max}|)$, where $S_{\max}$ is the largest set in $\mathcal{S}$ and H the harmonic function, defined as $H(d) = \sum_{i=1}^{d} 1/i$. For simplicity, assume each set has weight 1.

We use the following charging scheme: *when the algorithm adds a set S to the collection, let u denote the number of not-yet-covered elements in S and charge $1/u$ to each of those elements.* Clearly, the weight of the chosen sets is at most the total amount charged. To finish, we observe that the total amount charged is at most OPT × $H(|S_{\max}|)$. To see why this is so, let $S^* = \{e_s, e_{s-1}, \ldots, e_1\}$ be any set in OPT. Assume that when the greedy algorithm chooses sets to add to its collection, it covers the elements in S^* in the order given (each e_i is covered by the time e_{i-1} is). When the charge for an element e_i is computed (i.e., when the greedy algorithm chooses a set S containing e_i for the first time) at least i elements $(e_i, e_{i-1}, e_{i-2}, \ldots, e_1)$ in S^* are not yet covered. Since the greedily chosen set S contains at least as many not-yet-covered elements as S^*, the charge to e_i is at most $1/i$. Thus, the total charge to elements in S^* is at most

$$\frac{1}{s} + \frac{1}{s-1} + \cdots + \frac{1}{2} + 1 = H(s) \leq H(|S_{\max}|)$$

Thus, the total charge to elements covered by OPT is at most OPT × $H(|S_{\max}|)$. Since every element is covered by OPT, this means that the total charge is at most OPT × $H(|S_{\max}|)$. This implies that the greedy algorithm is an $H(|S_{\max}|)$-approximation algorithm.

These results were first reported in the mid-1970s [3–6]. Since then, it has been proven that no polynomial-time approximation algorithm for set cover has a significantly better approximation ratio unless P = NP [7].

The algorithm and approximation ratio extend to a fairly general class of problems called *minimizing a linear function subject to a submodular constraint.* This problem generalizes set cover as follows. Instead of asking for a set cover, we ask for a collection of sets C such that some function $f(C) \geq f(X)$. The function $f(C)$ should be increasing as we add sets to C and it should have the following property: if $C \subset C'$, then for any set S, $f(C' \cup \{S'\}) - f(C') \leq f(C \cup \{S\}) - f(C)$. In terms of the greedy algorithm, this means that adding a set S to the collection now increases f at least as much as adding it later. (For set cover, take $f(C)$ to be the number of elements covered by sets in C.) See Ref. [8] for details.

4.2.2 Shortest Superstring Problem

We consider an application of the set cover problem, SHORTEST SUPERSTRING problem. Given an alphabet Σ, and a collection of n strings $S = \{s_1, \ldots, s_n\}$, where each s_i is a string from the alphabet Σ, find a shortest string s that contains each s_i as a substring. There are several constant-factor approximation algorithms for this problem [9]; here we simply want to illustrate how to reduce this problem to the set cover problem. The reduction is such that an optimal solution to the set cover problem has weight at most twice the length of a shortest superstring.

For each $s_i, s_j \in S$ and for each value $0 < k < \min(|s_i|, |s_j|)$, we first check to see if the last k symbols of s_i are identical to the first k symbols of s_j. If so, we define a new string β_{ijk} obtained by concatenating s_i with s_j^k, the string obtained from s_j by deleting the first k characters of s_j. Let $\mathcal{C}$ be the set of strings β_{ijk}. For a string π we define $S(\pi) = \{s \in S | s \text{ is a substring of } \pi\}$. The underlying set of elements of the set cover is S. The specified subsets of S are the sets $S(\pi)$ for each $\pi \in S \cup \mathcal{C}$. The weight of each set $S(\pi)$ is $|\pi|$, the length of the string.

We can now apply the greedy set cover algorithm to find a collection of sets $S(\pi_i)$ and then simply concatenate the strings π_i to find a superstring. The approximation factor of this algorithm can be shown to be $2H(n)$.

4.3 Steiner Trees

The STEINER TREE problem is defined as follows. Given an edge-weighted graph $G = (V, E)$ and a set of terminals $S \subseteq V$, find a minimum-weight tree that includes all the nodes in S. (When $S = V$, then this is the problem of finding a minimum-weight *spanning* tree. There are several very fast greedy algorithms that can be used to solve this problem optimally.) The Steiner tree problem is NP-hard and several greedy algorithms have been designed that give a factor 2 approximation [10,11]. We briefly describe the idea behind one of the methods. Let $T_1 = \{s_1\}$ (an arbitrarily chosen terminal from S). At each step, T_{i+1} is computed from T_i as follows: attach the vertex from $S - T_i$ that is the "closest" to T_i by a path to T_i and call the newly added special vertex s_{i+1}. Thus T_i always contains the vertices $s_1, s_2, \ldots, s_i$. It is clear that the solution produces a Steiner tree. It is possible to prove that the weight of this tree is at most twice the weight of an optimal Steiner tree.

Zelikovsky [12] developed a greedy algorithm with an approximation ratio of 11/6. This bound has been further improved subsequently, but by using more complex methods.

A generalization of Steiner trees called NODE-WEIGHTED STEINER TREES is defined as follows. Given a node-weighted graph $G = (V, E)$ and a set of terminals $S \subset V$, find a minimum-weight tree that includes all the nodes in S. Here, the weight of a tree is the sum of the weights of its nodes. It can be shown that this problem is at least as hard as the set cover problem to approximate [13]. Interestingly, this problem is solved via a greedy algorithm similar to the one for the set cover problem with costs. We define a "spider" as a tree on ℓ terminals, where there is at most one vertex with degree more than 2. Each leaf in the tree corresponds to a terminal. The weight of the spider is simply the weight of the nodes in the spider. The algorithm at each step greedily picks a spider with minimum ratio of weight to number of terminals in it. It collapses all the terminals spanned by the spider into a single vertex, makes this new vertex a terminal and repeats until one terminal remains. The approximation guarantee of this algorithm is $2 \ln |S|$. Further improvements appear in Ref. [14]. For more on the Steiner tree problem, see the book by Hwang et al. [15].

4.4 *K*-Centers

The K-CENTER problem is a fundamental facility location problem and is defined as follows: given an edge-weighted graph $G = (V, E)$, find a subset $S \subseteq V$ of size at most K such that each vertex in V is close to some vertex in S. More formally, the objective function is defined as follows:

$$\min_{S \subseteq V} \max_{u \in V} \min_{v \in S} d(u, v)$$

where d is the distance function. For example, one may wish to install K fire stations and minimize the maximum distance (response time) from a location to its closest fire station.

Gonzalez [16] describes a very simple greedy algorithm for the basic K-center problem and proves that it gives an approximation factor of 2. The algorithm works as follows. Initially, pick any node v_0 as a center and add it to the set C. Then for $i = 1$ to K do the following: in iteration i, for every node $v \in V$, compute its distance $d^i(v, C) = \min_{c \in C} d(v, c)$ to the set C. Let v_i be a node that is farthest away from C, i.e., a node for which $d^i(v_i, C) = \max_{v \in V} d(v, C)$. Add v_i to C. Return the nodes $v_0, v_1, \ldots, v_{K-1}$ as the solution.

The above greedy algorithm is a 2-approximation for the K-center problem. First note that the radius of our solution is $d^K(v_K, C)$, since by definition v_K is the node that is farthest away from our set of centers. Now consider the set of nodes $v_0, v_1, \ldots, v_K$. Since this set has cardinality $K + 1$, at least two of these nodes, say v_i and v_j, must be covered by the same center c in the optimal solution. Assume without loss of generality that $i < j$. Let R^* denote the radius of the optimal solution. Observe that the distance from each node to the set C does not increase as the algorithm progresses. Therefore $d^K(v_K, C) \leq d^j(v_K, C)$. Also we must have $d^j(v_K, C) \leq d^j(v_j, C)$, otherwise we would not have selected node v_j in iteration j. Therefore,

$$d(c, v_i) + d(c, v_j) \geq d(v_i, v_j) \geq d^j(v_j, C) \geq d^K(v_K, C)$$

by the triangle inequality and the fact that v_i is in the set C at iteration j. But since $d(c, v_i)$ and $d(c, v_j)$ are both at most R^*, we have the radius of our solution $= d^K(v_K, C) \leq 2R^*$.

4.5 Connected Dominating Sets

The connected dominating set (CDS) problem is defined as follows. Given a graph $G = (V, E)$, find a minimum size subset S of vertices, such that the subgraph induced by S is connected and S forms a dominating set in G. Recall that a dominating set is one in which each vertex is either in the dominating set or adjacent to some vertex in the dominating set. The CDS problem is known to be *NP*-hard.

We describe a greedy algorithm for this problem [17]. The algorithm runs in two phases. At the start of the first phase all nodes are colored white. Each time we include a vertex in the dominating set, we color it black. Nodes that are dominated are colored gray (once they are adjacent to a black node). In the first phase, the algorithm picks a node at each step and colors it black, coloring all adjacent white nodes gray. A *piece* is defined as a white node or a black connected component. *At each step we pick a node to color black that gives the maximum (nonzero) reduction in the number of pieces.*

It is easy to show that at the end of this phase if no vertex gives a nonzero reduction to the number of pieces, then there are no white nodes left.

In the second phase, we have a collection of black connected components that we need to connect. Recursively, connect pairs of black components by choosing a chain of vertices, until there is one black connected component. Our final solution is the set of black vertices that form the connected component.

Key Property: At the end of the first phase if there is more than one black component, then there is always a pair of black components that can be connected by choosing a chain of two vertices.

It can be shown that the CDS found by the algorithm is of size at most $(\ln \Delta + 3) \cdot |OPT_{CDS}|$, where Δ is the maximum degree of a node.

Let a_i be the number of pieces left after the *i*th iteration, and $a_0 = n$. Since a node can connect up to Δ pieces, $|OPT_{CDS}| \geq \frac{a_0}{\Delta}$. (This is true if the optimal solution has at least two nodes.) Consider the $(i+1)^{\text{th}}$ iteration. An optimal solution can connect a_i pieces. Hence, the greedy procedure is guaranteed to pick a node which connects at least $\lceil \frac{a_i}{|OPT_{CDS}|} \rceil$ pieces. Thus, the number of pieces will reduce by at least $\lceil \frac{a_i}{|OPT_{CDS}|} \rceil - 1$. This gives us the recurrence relation

$$a_{i+1} \leq a_i - \left\lceil \frac{a_i}{|OPT_{CDS}|} \right\rceil + 1 \leq a_i \left(1 - \frac{1}{|OPT_{CDS}|}\right) + 1$$

Its solution is

$$a_{i+1} \leq a_0 \left(1 - \frac{1}{|OPT_{CDS}|}\right)^i + \sum_{j=i}^{i-1} \left(1 - \frac{1}{|OPT_{CDS}|}\right)^j$$

Notice after $|OPT_{CDS}| \ln \frac{a_0}{|OPT_{CDS}|}$ iterations, the number of pieces left is less than $2|OPT_{CDS}|$. After this, for each node we choose, we will decrease the number of pieces by at least one until the number of black components is at most $|OPT_{CDS}|$, thus at most $|OPT_{CDS}|$ more vertices are picked. So after $|OPT_{CDS}| \ln \frac{a_0}{|OPT_{CDS}|} + |OPT_{CDS}|$ iterations at most $|OPT_{CDS}|$ pieces are left to connect. We connect the remaining pieces choosing chains of at most two vertices in the second phase. The total number of nodes chosen is at most $|OPT_{CDS}| \ln \frac{a_0}{|OPT_{CDS}|} + |OPT_{CDS}| + 2|OPT_{CDS}|$, and since $\Delta \geq \frac{a_0}{|OPT_{CDS}|}$, the solution found has at most $|OPT_{CDS}|(\ln \Delta + 3)$ nodes.

4.6 Scheduling

We consider the following simple scheduling problem [18]. There are k identical machines. We are given a collection of n jobs. Job J_i is specified by the vector: (r_i, d_i, p_i, w_i). The job has a *release* time of r_i, a *deadline* of d_i and a *processing time* of p_i. The weight of the job is w_i. Our goal is to schedule a subset of the jobs such that each job starts after its release time and is completed by its deadline. If S is a subset of jobs that are scheduled, then the total profit due to set S is $\sum_{J_i \in S} w_i$. We do not get any profit if the job is not completed by its deadline. Our objective is to find a maximum-profit subset of jobs that can be scheduled on the k machines. The jobs are scheduled on one machine, with no preemption. In other words, if job J_i starts on machine j at time s_i, then $r_i \leq s_i$ and $s_i + p_i \leq d_i$. Moreover, each machine can be executing at most one job at any point of time.

A number of algorithms for the problem are based on linear program (LP) rounding [18]. A special case of interest is when all jobs have unit weight (or identical weight). In this case, we simply wish to maximize the number of scheduled jobs. The following greedy algorithm has the property that it schedules a set of jobs such that the total number of scheduled jobs is at least ρ_k times the number of jobs in an optimal schedule. Here $\rho_k = 1 - \frac{1}{(1+\frac{1}{k})^k}$. Observe that when $k = 1$, then $\rho_k = \frac{1}{2}$, and this bound is tight for the greedy algorithm.

The algorithm considers each machine in turn and finds a maximal set of jobs to schedule for the machine; it removes these jobs from the collection of remaining jobs, then recurses on the remaining set of jobs. Now we discuss how a maximal set of jobs is chosen for a single machine. The idea is to pick a job that can be finished as quickly as possible. After we pick this job, we schedule it, starting it at the earliest possible time. Making this choice might force us to reject several other jobs. We then consider starting a job after the end of the last scheduled job, and again pick one that we can finish at the earliest possible time. In this way, we construct the schedule for a single machine.

4.7 Minimum-Degree Spanning Trees

In this problem, the input is a graph $G = (V, E)$, with nonnegative weights $w : E \mapsto R^+$ on its edges. We are also given an integer $d > 1$. The objective of the problem is to find a minimum-weight spanning tree of G in which the degree of every node is at most d. It is a generalization of the Hamiltonian path problem, and is therefore NP-hard. It is known that the problem is not approximable to any ratio unless $P = NP$ or the approximation algorithm is allowed to output a tree whose degree is greater than d. Approximation algorithms try to find a tree whose degree is as close to d as possible, but whose weight is not much more than an optimal degree-d tree.

Greedy algorithms usually select one edge at a time, and once an edge is chosen, that decision is never revoked and the edge is part of the output. Here we add a subset S of edges at a time (e.g., a spanning forest), where S is chosen to minimize a relaxed version of the objective function. We get an iterative solution and the output is a union of the edges selected in each of the steps. This approach typically provides a logarithmic approximation. For minimum-degree spanning trees (MDST), the algorithm finds a tree of degree $O(d \log n)$, whose weight is within $O(\log n)$ of an optimal degree-d tree, where the graph has n vertices. The ideas have appeared in Refs. [19,20]. Such algorithms in which two objectives (degree and weight) are approximated are called *bicriteria* approximation algorithms.

A minimum-weight subgraph in which each node has degree at most d and at least 1 can be computed using algorithms for matching. Except for possibly being disconnected, this subgraph satisfies the other properties of an MDST: degree constraints and weight at most OPT. A greedy algorithm for MDST works by repeatedly finding d-forests, where each d-forest is chosen to connect the connected components left from the previous stages. The number of components decreases by a constant factor in each stage, and, in $O(\log n)$ stages, we get a tree of degree at most $d \log n$.

4.8 Maximum-Weight b-Matchings

In this problem, we are interested in computing a maximum-weight subgraph of a given graph G in which each node has degree at most b. The classical matching problem is a b-matching with $b = 1$. This problem can be solved optimally in polynomial time, but the algorithms take about $O(n^3)$ time. We discuss a 1/2-approximation algorithm that runs in $O(bE + E \log E)$ time. The edges are sorted by weight, with the heaviest edges considered first. Start with an empty forest as the initial solution. When an edge is considered, we see if adding it to the solution violates the degree bound of its end vertices. If not, we add it to our solution. Intuitively, each edge of our solution can displace at most 2 edges of an optimal solution, one incident to each of its end vertices, but of lesser weight.

4.9 Primal-Dual Methods

In this section we study a powerful technique, namely the primal-dual method, for designing approximation algorithms [21]. Duality provides a systematic approach for bounding OPT, a key task in proving any approximation ratio. The approach underlies many approximation algorithms. In this section, we illustrate the basic method via a simple example.

A closely related method, one that we do not explore here, is the "local-ratio" method developed by Bar-Yehuda [22]. It seems that most problems that have been solved by the primal-dual method, appear amenable to attack by the local-ratio method as well.

We use as our example another fundamental NP-hard problem, the VERTEX COVER problem. Given a graph $G = (V, E)$ with weights on the vertices given by $w(v)$, we wish to find a minimum-weight vertex cover. A vertex cover is a subset of vertices, $S \subseteq V$, such that for each edge $(u, v) \in E$, either $u \in S$ or $v \in S$ or both. This problem is equivalent to the special case of the set cover problem, where each set contains exactly two elements.

We describe a 2-approximation algorithm. First, write an integer linear program (ILP) for this problem. For each vertex v in the given graph, the program has a binary variable $x_v \in \{0, 1\}$. Over this space of variables, the problem is to find

$$\min \Big\{ \sum_{v \in V} w(v) x_v : x_u + x_v \geq 1 \quad (\forall (u, v) \in E) \Big\}$$

It is easy to see that an optimal solution to this integer program gives an optimal solution to the original vertex cover problem. Thus, the integer program is NP-hard to solve. Instead of solving it directly, we relax ILP to an LP, which is to optimize the same objective function over the same set of constraints, but with real-valued variables $x_v \in [0, 1]$.

Each LP has a dual. Let $N(v)$ denote the neighbor set of v. The dual of LP has a variable $y_{(u,v)} \geq 0$ for each edge $(u, v) \in E$. Over this space of variables, the dual of LP is to find

$$\max \Big\{ \sum_{(u,v) \in E} y_{u,v} : \sum_{u \in N(v)} y_{(u,v)} \leq w(v) \quad (\forall v \in V) \Big\}$$

The key properties of these programs are the following:

1. Weak duality: The cost of any feasible solution to the dual is a lower bound on the cost of any feasible solution to LP. Consequently, the cost of any feasible solution to the dual is a lower bound on the cost of any feasible solution to ILP.
2. If we can find feasible solutions for ILP and the dual, where the cost of our solution to ILP is at most α times the cost of our solution to the dual, then our solution to ILP has cost at most α OPT.

One way to get an approximate solution is to solve the vertex cover LP optimally (e.g., using a network flow algorithm [23]), and then round the obtained fractional solution to an integral solution. Here we describe a different algorithm—a greedy algorithm that computes solutions to both ILP and the dual. The solutions are not necessarily optimal, but will have costs within a factor of 2.

The dual solution is obtained by the following simple heuristic: *Initialize all dual variables to 0, then simultaneously and uniformly raise all dual variables, except those dual variables that occur in constraints that are currently tight. Stop when all constraints are tight.* The solution to ILP is obtained as follows: *Compute the dual solution above. When the constraint for a vertex v becomes tight, add v to the cover.* (Thus, the vertices in the cover are those whose constraints are tight.)

The constraint for vertex v is tight if $\sum_{u\in N(v)} y_{(u,v)} = w(v)$. When we start to raise the dual variables, the sum increases at a rate equal to the degree of the vertex. Thus, the first vertices to be added are those minimizing $\frac{w(v)}{d(v)}$. These vertices and their edges are effectively deleted from the graph, and the process continues.

The algorithm returns a vertex cover because, in the end, for each edge (u, v) at least one of the two vertex constraints is tight. By weak duality, to see that the cost of the cover is at most 2OPT, it suffices to see that the cost of the cover S is at most twice the cost of the dual solution. This is true because each node's weight can be charged to the dual variables corresponding to the incident edges, and each such dual variable is charged at most twice:

$$\sum_{v\in S} w(v) = \sum_{v\in S}\sum_{u\in N(v)} y_{(u,v)} \le 2 \sum_{(u,v)\in E} y_{(u,v)}$$

The equality above follows because $w(v) = \sum_{u\in N(v)} y_{(u,v)}$ for each vertex added to the cover. The inequality follows because each dual variable $y_{(u,v)}$ occurs at most twice in the sum.

To implement the algorithm, it suffices to keep track of the current degree $D(v)$ of each vertex v as well as the slack $W(v)$ remaining in the constraint for v. In fact, with a little bit of effort the reader can see that the following pseudocode implements the algorithm described above, without explicitly keeping track of dual variables. This algorithm was first described by Clarkson [24]:

```
GREEDY-VERTEX-COVER($G$, $S$)
for all $v \in V$ do $W(v) \leftarrow w(v)$; $D(v) \leftarrow deg(v)$
$S \leftarrow \emptyset$
while $E \neq \emptyset$ do
    Find $v \in V$ for which $\frac{W(v)}{D(v)}$ is minimized.
    for all $u \in N(v)$ do
        $E \leftarrow E \setminus (u, v)$
        $W(u) \leftarrow W(u) - \frac{W(v)}{D(v)}$ and $D(u) \leftarrow D(u) - 1$
    end
    $S \leftarrow S \cup \{v\}$ and $V \leftarrow V \setminus \{v\}$
end
```

More sophisticated applications of the primal-dual method require more sophisticated proofs. In some cases, the algorithm starts with a greedy phase, but then has a final round in which some previously added elements are discarded. The key idea is to develop the primal solution hand in hand with the dual solution in a way that allows the cost of the primal solution to be "charged" to the cost of the dual.

Because the vertex cover problem is a special case of the set cover problem, it is also possible to solve the problem using the greedy set cover algorithm. This gives an approximation ratio of at most $H(|V|)$, and in fact there are vertex cover instances for which that greedy algorithm produces a solution of cost $\Omega(H(|V|))$ OPT. The greedy algorithm described above is almost the same; it differs only in that it modifies the weights of the neighbors of the chosen vertices as it proceeds. This slight modification yields a significantly better approximation ratio.

4.10 Greedy Algorithms via the Probabilistic Method

In their book on the probabilistic method, Alon et al. [25] describe probabilistic proofs as follows:

> In order to prove the existence of a combinatorial structure with certain properties, we construct an appropriate probability space and show that a randomly chosen element in the space has the desired properties with positive probability.

The *method of conditional probabilities* is used to convert those proofs into efficient algorithms [26].

For some problems, elementary probabilistic arguments easily prove that good solutions exist. In some cases (especially when the proofs are based on iterated random sampling), the probabilistic proof can be converted into a greedy algorithm. This is a fairly general approach for designing greedy algorithms. In this section we give some examples.

4.10.1 Max Cut

Given a graph $G = (V, E)$, the MAX-CUT problem is to partition the vertices into two sets S and $\overline{S}$ so as to maximize the number of edges "cut" (crossing between the two sets). The problem is NP-hard.

Consider the following randomized algorithm: *For each vertex, choose the vertex to be in S or $\overline{S}$ independently with probability* 1/2. We claim this is a 1/2-approximation algorithm, in expectation. To see why, note that the probability when any given edge is cut is 1/2. Thus, by linearity of expectation, in expectation $|E|/2$ edges are cut. Clearly an optimal solution cuts at most twice this many edges.

Next, we apply the *method of conditional probabilities* [25,26] to convert this randomized algorithm into a deterministic one. We replace each random choice made by the algorithm by a deterministic choice that does "as well" in a precise sense. Specifically, we modify the algorithm to maintain the following invariant:

> *After each step, if we were to take the remaining choices randomly, then the expected number of edges cut in the end would be at least* |E|/2*.*

Suppose decisions have been made for vertices $V_t = \{v_1, v_2, \ldots, v_t\}$, but not yet for vertex v_{t+1}. Let S_t denote the vertices in V_t chosen to be in S. Let $\overline{S}_t = V_t - S_t$ denote the vertices in V_t chosen to be in $\overline{S}$. Given these decisions, the status of each edge in $V_t \times V_t$ is known, while the rest still have a 1/2 probability of being cut. Let $x_t = |E \cap (S_t \times \overline{S}_t)|$ denote the number of those edges that will definitely cross the cut. Let $e_t = |E - V_t \times V_t|$ denote the number of edges which are not yet determined. Then, given the decisions made so far, the expected number of edges that would be cut if all remaining choices were to be taken randomly would be

$$\phi_t \doteq x_t + e_t/2$$

The x_t term counts the edges cut so far, while the $e_t/2$ term counts the e_t edges with at least one undecided endpoint: each of those edges will be cut with probability 1/2.

Our goal is to replace the random decisions for the vertices with deterministic decisions that guarantee $\phi_{t+1} \geq \phi_t$ at each step. If we can do this, then we will have $|E|/2 = \phi_0 \leq \phi_1 \leq \cdots \leq \phi_n$, and, since ϕ_n is the number of edges finally cut, this will ensure that at least $|E|/2$ edges are cut.

Consider deciding whether the vertex v_{t+1} goes into S_{t+1} or $\overline{S}_{t+1}$. Let s be the number of v_{t+1}'s neighbors in S_t. Let $\overline{s}$ be the number of v_{t+1}'s neighbors in $\overline{S}_{t+1}$. By calculation

$$\phi_{t+1} - \phi_t = \begin{cases} s/2 - \overline{s}/2 & \text{if } v_{t+1} \text{ is added to } \overline{S}_{t+1} \\ \overline{s}/2 - s/2 & \text{otherwise} \end{cases}$$

Thus, the following strategy ensures $\phi_{t+1} \geq \phi_t$: *if $s \leq \overline{s}$, then put v_{t+1} in S_{t+1}; otherwise put v_t in $\overline{S}_{t+1}$*. By doing this at each step, the algorithm guarantees that $\phi_n \geq \phi_{n-1} \geq \cdots \geq |E|/2$.

We have derived the following greedy algorithm: *Start with $S = \overline{S} = \emptyset$. Consider the vertices in turn. For each vertex v, put the vertex v in S or $\overline{S}$, whichever has fewer of v's neighbors.* We know from the derivation that this is a 1/2-approximation algorithm.

4.10.2 Independent Set

Although the application of the method of conditional probabilities is somewhat technical, it is routine, in the sense that it follows a similar form in every case. Here is another example.

The problem of finding a MAXIMUM INDEPENDENT SET in a graph $G = (V, E)$ is one of the most basic problems in graph theory. An independent set is defined as a subset S of vertices such that there are no edges between any pair of vertices in S. The problem is NP-hard. Turan's theorem states the following: *Any graph G with n nodes and average degree d has an independent set I of size at least $n/(d+1)$*. Next, we sketch a classic proof of the theorem using the probabilistic method. Then we apply the method of conditional probabilities to derive a greedy algorithm.

Let $\hat{N}(v) = N(v) \cup \{v\}$ denote the neighbor set of v, including v. Consider this randomized algorithm: *Start with $I = \emptyset$. Consider the vertices in a random order. When considering v, add it to I if $\hat{N}(v) \cap I = \emptyset$.*

For a vertex v to be added to I, it suffices for v to be considered before any of its neighbors. This happens with probability $|\hat{N}(v)|^{-1}$. Thus, by linearity of expectation, the expected number of vertices added to I is at least

$$\sum_{v} |\hat{N}(v)|^{-1}$$

A standard convexity argument shows this is at least $n/(d+1)$, completing the proof of Turan's theorem.

Now we apply the method of conditional probabilities. Suppose the first t vertices $V_t = \{v_1, v_2, \ldots, v_t\}$ have been considered. Let $I_t = V_t \cap I$ denote those that have been added to I. Let $R_t = V \setminus (V_t \cup \hat{N}(I_t))$ denote the remaining vertices that might still be added to I and let $\hat{N}_t(v) = \hat{N}(v) \cap R_t$ denote the neighbors of v that might still be added. If the remaining vertices were to be chosen in random order, the expected number of vertices in I by the end would be at least

$$\phi_t \doteq |I_t| + \sum_{v \in R_t} |\hat{N}_t(v)|^{-1}$$

We want the algorithm to choose vertex v_{t+1} to ensure $\phi_{t+1} \geq \phi_t$. To do this, it suffices to choose the vertex $w \in R_t$ minimizing $|\hat{N}_t(w)|$, for then

$$\phi_{t+1} - \phi_t \geq 1 - \sum_{v \in \hat{N}_t(w)} |\hat{N}_t(v)|^{-1} \geq 1 - \sum_{v \in \hat{N}_t(w)} |\hat{N}_t(w)|^{-1} = 0$$

This gives us the following greedy algorithm: *Start with $I = \emptyset$. Repeat until no vertices remain: Choose a vertex v of minimum degree in the remaining graph; add v to I and delete v and all of its neighbors from the graph. Finally, return I.* It follows from the derivation that this algorithm ensures $n/(d+1) \leq \phi_0 \leq \phi_1 \leq \cdots \leq \phi_n$, so that the algorithm returns an independent set of size at least $n/(d+1)$, where d is the average degree of the graph.

As an exercise, the reader can give a different derivation leading to the following greedy algorithm (with the same performance guarantee): *Order the vertices by increasing degree, breaking ties arbitrarily. Let I consist of those vertices that precede all their neighbors in the ordering.*

4.10.3 Unweighted Set Cover

Next we illustrate the method on the set cover problem.

We start with a randomized rounding scheme that uses iterated random sampling to round a fractional set cover (a solution to the relaxed problem) to a true set cover. We prove an approximation ratio for the randomized algorithm, then apply the method of conditional probabilities to derive a deterministic greedy algorithm.

We emphasize that, in applying the method of conditional probabilities, we remove the explicit dependence of the algorithm on the fractional set solution. Thus, the final algorithm does not in fact require solving the relaxed problem first.

Recall the definition of the set cover problem from the beginning of the chapter. For this section, we will assume all weights $w(S_i)$ are 1.

Consider the following relaxation of the problem: assign a value $z_i \in [0, 1]$ to each set S_i so as to minimize $\sum_i z_i$ subject to the constraint that, for every element x_j, $\sum_{i:x_j \in S_i} z_i \geq 1$. We call a z meeting these constraints a *fractional set cover*.

The optimal set cover gives one possible solution to the relaxed problem, but there may be other fractional set covers that give a smaller objective function value. However, not too much smaller. We claim the following: *Let z be any fractional set cover. Then there exists an actual set cover C of size at most $T = \lceil \ln(n)|z| \rceil$, where $|z| = \sum_i z_i$.*

To prove this, consider the following randomized algorithm: given z, *draw T sets at random from the distribution p defined by $p(S_i) = z_i/|z|$*. With nonzero probability, this random experiment yields a set cover. Here is why. A calculation shows that, with each draw, the chance that any given element e is covered is at least $1/|z|$. Thus, the expected number of elements left uncovered after T draws is at most

$$n(1 - 1/|z|)^T \; < \; n \exp(-T/|z|) \; \leq \; 1$$

Since on average less than one element is left uncovered, it must be that some outcome of the random experiment covers all elements.

Next we apply the method of conditional probabilities. Suppose that t sets have been chosen so far, and let n_t denote the number of elements not yet covered. Then the conditional expectation of the number of elements left uncovered at the end is at most

$$\phi_t \; \doteq \; n_t(1 - 1/|z|)^{T-t}$$

We want the algorithm to choose each set to ensure $\phi_t \leq \phi_{t-1}$, so that in the end $\phi_T \leq \phi_0 < 1$ and the chosen sets form a cover.

Suppose the first t sets have been chosen, so that ϕ_t is known. A calculation shows that, if the next set is chosen at random according to the distribution p, then $E[\phi_{t+1}] \leq \phi_t$. Thus, choosing the next set to *minimize* ϕ_{t+1} will ensure $\phi_{t+1} \leq \phi_t$. By inspection, choosing the set to minimize ϕ_{t+1} is the same as choosing the set to minimize n_{t+1}.

We have derived the following greedy algorithm: *Repeat T times: add a set to the collection so as to minimize the number of elements remaining uncovered.* In fact, it suffices to do the following: *Repeat until all elements are covered: add a set to the collection so as to minimize the number of elements remaining uncovered.* (This suffices because we know from the derivation that a cover will be found within T rounds.)

We have proven the following fact: *The above greedy algorithm returns a cover of size at most* $\min_z \lceil \ln(n)|z| \rceil$, *where z ranges over all fractional set covers.* Since the minimum-size set cover OPT corresponds to a z with $|z| = |\text{OPT}|$, we have the following corollary: *The above greedy algorithm returns a cover of size at most* $\lceil \ln(n)\text{OPT} \rceil$.

This algorithm can be generalized to weighted set cover, and slightly stronger performance guarantees can be shown [3–6]. This particular greedy approach applies to a general class of problems called "minimizing a linear function subject to a submodular constraint" [8].

Comment: In many cases, applying the method of conditional probabilities will not yield a greedy algorithm, because the conditional expectation ϕ_t will depend on the fractional solution in a nontrivial way. In that case, the derandomized algorithm will first have to compute the fractional solution (typically by solving a linear program). That is Raghavan and Thompson's standard method of *randomized rounding* [27]. The variant we see here was first observed in Ref. [28]. Roughly, to get a greedy algorithm, we should apply the method of conditional probabilities to a probabilistic proof based on *repeated random sampling* from the distribution defined by the fractional optimum.

4.10.4 Lagrangian Relaxation for Fractional Set Cover

The algorithms described above fall naturally into a larger and technically more complicated class of algorithms called *Lagrangian relaxation algorithms*. Typically, such an algorithm is used to find a structure meeting a given set of constraints. The algorithm constructs a solution in small steps. Each step is made so as to minimize (or keep from increasing) a *penalty function* that approximates some of the underlying constraints. Finally, the algorithm returns a solution that approximately meets the underlying constraints.

These algorithms typically have a greedy outer loop. In each iteration, they solve a subproblem that is simpler than the original problem. For example, a multicommodity flow algorithm may solve a sequence of shortest-path subproblems, routing small amounts of flow along paths chosen to minimize the sum of edge penalties that grow exponentially with the current flow on the edge.

Historical examples include algorithms by von Neumann, Ford and Fulkerson, Dantzig-Wolfe decomposition, Benders' decomposition, and Held and Karp. In 1990, Shahrokhi and Matula proved a polynomial time bound for such an algorithm for multicommodity flow. This sparked a long line of work generalizing and strengthening this result (e.g., [29–31]). See the recent text by Bienstock [32]. These works focus mainly on *packing and covering problems*—LPs and ILPs with nonnegative coefficients.

As a rule, the problems in question can also be solved by standard linear programming algorithms such as the simplex, the ellipsoid, or interior-point algorithms. The primary motivation for studying Lagrangian relaxation algorithms has been that, like other greedy algorithms, they can often be implemented without explicitly constructing the full underlying problem. This can make them substantially faster.

As an example, here is a Lagrangian relaxation algorithm for fractional set cover (given an instance of the set cover problem, find a fractional set cover z of minimum size $|z| = \sum_i z_i$; see the previous subsection for definitions). Given a set cover instance and $\varepsilon \in [0, 1/2]$, the algorithm returns a fractional set cover of size at most $1 + O(\varepsilon)$ times the optimum:

1. Let $N = 2\ln(n)/\varepsilon^2$, where n is the number of elements.
2. Repeat until all elements are sufficiently covered ($\min_j c(j) \geq N$).
3. Choose a set S_i maximizing $\sum_{x_j \in S_i} (1-\varepsilon)^{c(j)}$, where $c(j)$ denotes the number of times any set containing element x_j has been chosen so far.
4. Return z, where z_i is the number of times S_i was chosen divided by N.

The naive implementation of this algorithm runs in $O(nM\log(n)/\varepsilon^2)$ time, where $M = \sum_i |S_i|$ is the size of the input. With appropriate modifications, the algorithm can be implemented to run in $O(M\log(n)/\varepsilon^2)$ time.

For readers who are interested, we sketch how this algorithm may be derived using the probabilistic framework. To begin, we imagine that we have in hand any fractional set cover z^*, to which we apply the following randomized algorithm: *Define probability distribution p on the sets by $p(S_i) = z^*_i/|z^*|$. Draw sets randomly according to p until every element has been covered (in a drawn set) at least $N = 2\ln(n)/\varepsilon^2$ times. Return z, where z_i is the number of times set S_i was drawn, divided by N.* (The reader should keep in mind that the dependence on z^* will be removed when we apply the method of conditional probabilities.)

Claim: *With nonzero probability, the algorithm returns a fractional set cover of size at most* $(1 + O(\varepsilon))|z^*|$.

Next we prove the claim. Let $T = |z^*|N/(1-\varepsilon)$. We will prove that, with nonzero probability, within T draws each set will be covered at least N times. This will prove the claim because then the size of z is at most $T/N = |z^*|/(1-\varepsilon)$.

Fix a given element x_j. With each draw, the chance that x_j is covered is at least $1/|z^*|$. Thus, the expected number of times x_j is covered in T draws is at least $T/|z^*| = N/(1-\varepsilon)$. By a standard Chernoff bound, the probability that x_j is covered less than N times in T rounds is at most $\exp(-\varepsilon^2 N/2(1-\varepsilon)) < 1/n$.

By linearity of expectation, the expected number of elements that are covered less than N times in T rounds is less than 1. Thus, with nonzero probability, all elements are covered at least N times in T rounds.

This proves the claim. Next we apply the method of conditional probabilities to derive a greedy algorithm.

Let X_{jt} be an indicator variable for the event that x_j is covered in round t, so that for any j the X_{jt}'s are independent with $E[X_{jt}] \geq 1/|z^*|$. Let $\mu = N/(1-\varepsilon)$. The proof of the Chernoff-bound bounds $\Pr[\sum_t X_{jt} \leq (1-\varepsilon)\mu]$ by the expectation of the following quantity:

$$\frac{(1-\varepsilon)^{\sum_t X_{jt}}}{(1-\varepsilon)^{(1-\varepsilon)\mu}} = \frac{(1-\varepsilon)^{\sum_t X_{jt}}}{(1-\varepsilon)^N}$$

Thus, the proof of our claim above implicitly bounds the probability of failure by the expectation of

$$\phi = \sum_j \frac{(1-\varepsilon)^{\sum_t X_{jt}}}{(1-\varepsilon)^N}$$

Furthermore, the proof shows that the expectation of this quantity is less than 1.

To apply the method of conditional probabilities, we will *choose each set to keep the conditional expectation of the above quantity ϕ below 1.*

After the first t sets have been drawn, the random variables X_{js} for $s \leq t$ are determined, while X_{js} for $s > t$ are not yet determined. Using the inequalities from the proof of the Chernoff bound, the conditional expectation of ϕ given the choices for the first t sets is at most

$$\phi_t \doteq \sum_j \frac{\prod_{s \leq t}(1-\varepsilon)^{X_{js}} \times \prod_{s>t}(1-\varepsilon/|z^*|)}{(1-\varepsilon)^N}$$

This quantity is initially less than 1, so it suffices to *choose each set to ensure* $\phi_{t+1} \leq \phi_t$. If the $t+1$st set is chosen randomly according to p, then $E[\phi_{t+1}] \leq \phi_t$. Thus, to ensure $\phi_{t+1} \leq \phi_t$, it suffices to choose the set to minimize ϕ_{t+1}. By a straightforward calculation, this is the same as choosing the set S_i to maximize $\sum_{x_j \in S_i}(1-\varepsilon)^{\Sigma_{s \leq t} X_{jt}}$. This gives us the algorithm in question (at the top of this section). From the derivation, we know the following fact: *The algorithm above returns a fractional set cover of size at most* $(1+O(\varepsilon)) \min_{z^*} |z^*|$, *where z^* ranges over all the fractional set covers.*

4.11 Conclusions

In this chapter we surveyed a collection of problems and described simple greedy algorithms for several of these problems. In several cases, the greedy algorithms described do not represent the state of the art for these problems. The reader is referred to other chapters in this handbook to read in more detail about the specific problems and the techniques that yield the best worst-case approximation guarantees. In many instances, the performance of greedy algorithms may be better than their worst-case bounds suggest. This and their simplicity make them important in practice.

For some problems (e.g., set cover), it is known that a greedy algorithm gives the best possible approximation ratio unless $NP \subset \mathrm{DTIME}(n^{\log\log n})$. But for some problems no such intractability results are yet known. In these cases, instead of proving hardness of approximation for all polynomial-time algorithms, one may try something easier: to prove that no *greedy* algorithm gives a good approximation. Of course this requires a formal definition of the class of algorithms. (A similar approach has been fruitful in competitive analysis of online algorithms.) Such a formal study of greedy algorithms with an eye toward lower bound results has been the subject of several recent papers [33].

For additional information on combinatorial optimization, the reader is referred to books by Papadimitriou and Steiglitz [2], Cook et al. [34], and a series of three books by Schrijver [35]. For more on approximation algorithms, there is a book by Vazirani [23], lecture notes by Motwani [36], and a book edited by Hochbaum [37]. There is a chapter on greedy algorithms in several textbooks, such as Kleinberg and Tardos [38], and Cormen et al. [39]. More on randomized algorithms can be found in a book by Motwani and Raghavan [40], and a survey by Shmoys [41].

References

[1] Lawler, E., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Wilson, New York, 1976.

[2] Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[3] Johnson, D. S., Approximation algorithms for combinatorial problems, *JCSS*, 9, 256, 1974.

[4] Stein, S. K., Two combinatorial covering theorems, *J. Comb. Theory A*, 16, 391, 1974.

[5] Lovász, L., On the ratio of optimal integral and fractional covers, *Discrete Math.*, 13, 383, 1975.

[6] Chvátal, V., A greedy heuristic for the set-covering problem, *Math. Oper. Res.*, 4(3), 233, 1979.

[7] Raz, R. and Safra, S., A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP, *Proc. of STOC*, 1997, p. 475.

[8] Nemhauser, G. L. and Wolsey, L. A., *Integer and Combinatorial Optimization*, Wiley, New York, 1988.

[9] Blum, A., Jiang, T., Li, M., Tromp, J., and Yannakakis, M., Linear approximation of shortest superstrings, *JACM*, 41(4), 630, 1994.

[10] Markowsky, G., Kou, L., and Berman, L., A fast algorithm for Steiner trees, *Acta Inform.*, 15, 141, 1981.

[11] Takahashi, H. and Matsuyama, A., An approximate solution for the Steiner problem in graphs, *Math. Japonica*, 24(6), 573, 1980.

[12] Zelikovsky, A., An 11/6-approximation algorithm for the network Steiner problem, *Algorithmica*, 9(5), 463, 1993.

[13] Klein, P. N. and Ravi, R., A nearly best-possible approximation algorithm for node-weighted Steiner trees, *J. Algorithms,* 19(1), 104, 1995.

[14] Guha, S. and Khuller, S., Improved methods for approximating node weighted Steiner trees and connected dominating sets, *Inf. Comput.*, 150(1), 57, 1999.

[15] Hwang, F. K., Richards, D. S., and Winter, P., *The Steiner Tree Problem*, Number 53 in Annals of Discrete Mathematics, Elsevier Science Publishers B. V., Amsterdam, 1992.

[16] Gonzalez, T. F., Clustering to minimize the maximum intercluster distance, *Theor. Comput. Sci.*, 38, 293, 1985.

[17] Guha, S. and Khuller, S., Approximation algorithms for connected dominating sets, *Algorithmica*, 20(4), 374, 1998.

[18] Bar-Noy, A., Guha, S., Naor, J., and Schieber, B., Approximating the throughput of multiple machines in real-time scheduling, *SIAM J. Comput.*, 31(2), 331, 2001.

[19] Fürer, M. and Raghavachari, B., An NC approximation algorithm for the minimum-degree spanning tree problem, *Proc. 28th Annu. Allerton Conf. on Communication, Control and Computing*, 1990, p. 274.

[20] Ravi, R., Marathe, M. V., Ravi, S. S., Rosenkrantz, D. J., and Hunt, H. B., III, Approximation algorithms for degree-constrained minimum-cost network-design problems, *Algorithmica*, 31(1), 58, 2001.

[21] Goemans, M. X. and Williamson, D. P., A general approximation technique for constrained forest problems, *SIAM J. Comput.*, 24(2), 296, 1995.

[22] Bar-Yehuda, R., One for the price of two: a unified approach for approximating covering problems, *Algorithmica*, 27(2), 131, 2000.

[23] Vazirani, V. V., *Approximation Algorithms*, Springer, New York, 2001.

[24] Clarkson, K., A modification of the greedy algorithm for vertex cover, *Inf. Process. Lett.,* 16, 23, 1983.

[25] Alon, N., Spencer, J. H., and Erdős, P., *The Probabilistic Method,* Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley, Chichester, 1992.

[26] Raghavan, P., Probabilistic construction of deterministic algorithms approximating packing integer programs, *JCSS*, 37(2), 130, 1988.

[27] Raghavan, P. and Thompson, C., Randomized rounding: a technique for provably good algorithms and algorithmic proofs, *Combinatorica*, 7, 365, 1987.

[28] Young, N. E., Randomized rounding without solving the linear program, *Proc. of SODA*, San Francisco, CA, 1995, p. 170.

[29] Plotkin, S. A., Shmoys, D. B., and Tardos, É., Fast approximation algorithms for fractional packing and covering problems, *Math. Oper. Res.*, 20(2), 257, 1995.

[30] Grigoriadis, M. D. and Khachiyan, L. G., Fast Approximation Schemes for Convex Programs with Many Blocks and Coupling Constraints, Technical Report DCS-TR-273, Rutgers University Computer Science Department, New Brunswick, NJ, 1991.

[31] Young, N. E., Sequential and parallel algorithms for mixed packing and covering, *Proc. IEEE FOCS*, 2001, p. 538.

[32] Bienstock, D., *Potential Function Methods for Approximately Solving Linear Programming Problems: Theory and Practice*, Kluwer Academic Publishers, Boston, MA, 2002.

[33] Borodin, A., Nielsen, M., and Rackoff, C., (Incremental) priority algorithms, *Algorithmica*, 37, 295, 2003.

[34] Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Schrijver, A., *Combinatorial Optimization,* Wiley, New York, 1997.

[35] Schrijver, A., *Combinatorial Optimization—Polyhedra and Efficiency, Volume A: Paths, Flows, Matchings, Volume B: Matroids, Trees, Stable Sets, Volume C: Disjoint Paths, Hypergraphs, Algorithms and Combinatorics*, Vol. 24, Springer, Berlin, 2003.

[36] Motwani, R., Lecture Notes on Approximation Algorithms, Technical Report, Stanford University, 1992.

[37] Hochbaum, D., Ed., *Approximation Algorithms for NP-Hard Problems,* PWS Publishing Company, Boston, MA, 1997.

[38] Kleinberg, J. and Tardos, É., *Algorithm Design*, Addison-Wesley, Reading, MA, 2005.

[39] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein C., *Introduction to Algorithms,* MIT Press, Cambridge, MA, 2001.

[40] Motwani, R. and Raghavan, P., *Randomized Algorithms*, Cambridge University Press, London, 1997.

[41] Shmoys, D. B., Computing near-optimal solutions to combinatorial optimization problems, in *Combinatorial Optimization,* Cook, W., Lovasz, L., and Seymour, P. D., Eds., AMS, Providence, RI, 1995, p. 355.

5
Recursive Greedy Methods

Guy Even
Tel Aviv University

5.1 Introduction

Greedy algorithms are often the first algorithms that one considers for various optimization problems, and, in particular, covering problems. The idea is very simple: try to build a solution incrementally by augmenting a partial solution. In each iteration, select the "best" augmentation according to a simple criterion. The term greedy is used because the most common criterion is to select an augmentation that minimizes the ratio of "cost" to "advantage." We refer to the cost-to-advantage ratio of an augmentation as the *density* of the augmentation.

In the set-cover (SC) problem, every set S has a weight (or cost) $w(S)$. The "advantage" of a set S with respect to a partial cover $\{S_1, \ldots, S_k\}$ is the number of new elements covered by S, i.e., $|S \setminus (S_1 \cdots S_k)|$. In each iteration, a set with a minimum density is selected and added to the partial solution until all the elements are covered. In the SC problem, it is easy to find an augmentation with minimum density simply by recomputing the density of every set in every iteration.

In this chapter, we consider problems for which it is NP-hard to find an augmentation with minimum density. From a covering point of view, this means that there are exponentially many sets. However, these sets are succinctly represented using a structure with polynomial complexity. For example, the sets can be paths or trees in a graph. In such problems, applying the greedy algorithm is a nontrivial task. One way to deal with such a difficulty is to try to approximate a minimum density augmentation. Interestingly, the augmentation itself is computed using a greedy algorithm, and this is why the algorithm is called the recursive greedy algorithm.

The recursive greedy algorithm was presented by Zelikovsky [1] and Kortsarz and Peleg [2]. In Ref. [1], the directed Steiner tree (DST) problem in acyclic graphs was considered. In the DST problem, the input consists of a directed graph $G = (V, E)$ with edge weights $w(e)$, a subset $X \subseteq V$ of terminals, and a root $r \in V$. The goal is to find a minimum-weight subgraph that contains directed paths from r to every terminal in X. In Ref. [2], the bounded diameter Steiner tree (BDST) problem was considered. In the BDST problem, the input consists of an undirected graph $G = (V, E)$ with edge costs $w(e)$, a subset of terminals $X \subseteq V$, and a diameter parameter d. The goal is to find a minimum-weight tree that spans

X with diameter bounded by d. In both papers, it is proved that, for every $\varepsilon > 0$, the recursive greedy algorithm achieves an $O(|X|^{\varepsilon})$ approximation ratio in polynomial time. The recursive greedy algorithm is still the only nontrivial approximation algorithm known for these problems.

The presentation of the recursive greedy algorithm was simplified and its analysis was perfected by Charikar et al. [3]. In Ref. [3], the recursive greedy algorithm was used for the DST problem. The improved analysis gave a poly-logarithmic approximation ratio in quasi-polynomial time (i.e., running time is $O(n^{c \log n})$, for a constant c).

The recursive greedy algorithm is a combinatorial algorithm (i.e., no linear programming or high precision arithmetic is used). The algorithm's description is simple and short. The analysis captures the intuition regarding the segments during which the greedy approach performs well. The running time of the algorithm is exponential in the depth of the recursion, and hence, reducing its running time is an important issue.

We present modifications of the recursive greedy algorithm that enable reducing the running time. Unfortunately, these modifications apply only to the restricted case in which the graph is a tree. We demonstrate these methods on the Group Steiner (GS) problem [4] and its restriction to trees [5]. Following Ref. [6], we show that for the GS problem over trees, the recursive greedy algorithm can be modified to give a poly-logarithmic approximation ratio in polynomial time. Better poly-logarithmic approximation algorithms were developed for the GS problem; however, these algorithms rely on linear programming [5,7].

5.1.1 Organization

In Section 5.2, we review the greedy algorithm for the SC problem. In Section 5.3, we present three versions of DST problems. We present simple reductions that allow us to focus on only one version. Section 5.4 constitutes the heart of this chapter; in it the recursive greedy algorithm and its analysis are presented. In Section 5.5, we consider the GS problem over trees. We outline modifications of the recursive greedy algorithm that enable a poly-logarithmic approximation ratio in polynomial time. We conclude in Section 5.6 with open problems.

5.2 A Review of the Greedy Algorithm

In this section we review the greedy algorithm for the SC problem and its analysis.

In the SC problem we are given a set of elements, denoted by $U = \{1, \ldots, n\}$ and a collection $\mathcal{R}$ of subsets of U. Each subset $S \in \mathcal{R}$ is also given a nonnegative weight $w(S)$. A subset $\mathcal{C} \subseteq \mathcal{R}$ is an SC if $\bigcup_{S' \in \mathcal{C}} S' = \{1, \ldots, n\}$. The *weight* of a subset of $\mathcal{R}$ is simply the sum of the weights of the sets in $\mathcal{R}$. The goal in the SC problem is to find a cover of minimum weight. We often refer to a subset of $\mathcal{R}$ that is not a cover as a *partial cover*.

The greedy algorithm starts with an empty partial cover. A cover is constructed by iteratively asking an *oracle* for a set to be added to the partial cover. This means that no backtracking takes place; every set that is added to the partial cover is kept until a cover is obtained. The oracle looks for a set with the lowest *residual density*, defined as follows.

Definition 5.1

Given a partial cover $\mathcal{C}$, the residual density of a set S is the ratio

$$\rho_{\mathcal{C}}(S) \triangleq \frac{w(S)}{|S \setminus \bigcup_{S' \in \mathcal{C}} S'|}$$

Note that the residual density is nondecreasing (and may even increase) as the greedy algorithm accumulates sets. The performance guarantee of the greedy algorithm is summarized in the following theorem (see Chapter 4).

Theorem 5.1

The greedy algorithm computes a cover whose cost is at most $(1 + \ln\ n) \cdot w(C^*)$, *where* C^* *is a minimum-weight cover.*

There are two main questions that we wish to ask about the greedy algorithm:

Question 1: What happens if the oracle is approximate? Namely, what if the oracle does not return a set with minimum residual density, but a set whose residual density is at most α times the minimum residual density? How does such an approximate oracle affect the approximation ratio of the greedy algorithm? In particular, we are interested in the case that α is not constant (e.g., α depends on the number of uncovered elements). We note that in the SC problem, an exact oracle is easy to implement. But we will see a generalization of the SC problem in which the task of an exact oracle is NP-hard, and hence we will need to consider an approximate oracle.

Question 2: What happens if we stop the execution of the greedy algorithm before a complete cover is obtained? Suppose that we stop the greedy algorithm when the partial cover covers $\beta \cdot n$ elements in U. Can we bound the weight of the partial cover? We note that one reason for stopping the greedy algorithm before it ends is that we simply run out of "budget" and cannot "pay" for additional sets.

The following lemma helps answer both questions. Let x denote the number of elements that are not covered by the partial cover. We say that the oracle is $\alpha(x)$-approximate if the residual density of the set it finds is at most $\alpha(x)$ times the minimum residual density.

Lemma 5.1 (Charikar et al. [3])

Suppose that the oracle of the greedy algorithm is $\alpha(x)$*-approximate and that* $\alpha(x)/x$ *is a nonincreasing function. Let* $\mathcal{C}_i$ *denote partial cover accumulated by the greedy algorithm after adding i sets. Then,*

$$\frac{w(\mathcal{C}_i)}{w(\mathcal{C}^*)} \leq \int_{n-|\cup_{S'\in\mathcal{C}_i} S'|}^{n} \frac{\alpha(x)}{x}\, dx$$

Proof

The proof is by induction on n. When $n = 1$, the algorithm simply returns a set S such that $w(S) \leq \alpha(1) \cdot w(\mathcal{C}^*)$. Since $\alpha(x)/x$ is nonincreasing, we conclude that $\alpha(1) \leq \int_0^1 \frac{\alpha(x)}{x}\, dx$, and the induction basis follows.

The induction step for $n > 1$ is proved as follows. Let $\mathcal{C}_i = \{S_1, \ldots, S_i\}$. When the oracle computes S_1, its density satisfies: $w(S_1)/|S_1| \leq \alpha(n) \cdot w(\mathcal{C}^*)/n$. Hence, $w(S_1) \leq |S_1| \cdot \frac{\alpha(n)}{n} \cdot w(\mathcal{C}^*)$. Since $\alpha(x)/x$ is nonincreasing, $|S_1| \cdot \frac{\alpha(n)}{n} \leq \int_{n-|S_1|}^{n} \frac{\alpha(x)}{x}\, dx$. We conclude that

$$w(S_1) \leq \int_{n-|S_1|}^{n} \frac{\alpha(x)}{x} dx \cdot w(\mathcal{C}^*) \tag{5.1}$$

Now consider the residual set system over the set of elements $\{1, \ldots, n\} \setminus S_1$ with the sets $S' = S \setminus S_1$. We keep the set weights unchanged, i.e., $w(S') = w(S)$. The collection $\{S'_2, \ldots, S'_i\}$ is the output of the greedy algorithm when given this residual set system. Let $n' = |S'_2 \cup \cdots \cup S'_i|$. Since $\mathcal{C}^*$ induces a cover of the residual set with the same weight as $w(\mathcal{C}^*)$, the induction hypothesis implies that

$$w(S'_2) + \cdots + w(S'_i) \leq \int_{n-(n'+|S_1|)}^{n-|S_1|} \frac{\alpha(x)}{x} dx \cdot w(\mathcal{C}^*). \tag{5.2}$$

The lemma follows now by adding Eq. (5.1) and Eq. (5.2). □

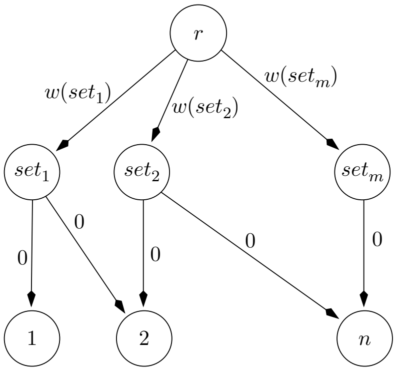


FIGURE 5.1 Reduction of SC instance to DST instance.

We remark that for a full cover, since $\int_0^1 dx/x$ is not bounded, one could bound the ratio by $\alpha(1) + \int_1^n \frac{\alpha(x)}{x}\,dx$. Note that for an exact oracle $\alpha(x) = 1$, this modification of Lemma 5.1 implies Theorem 5.1.

Lemma 5.1 shows that the greedy algorithm also works with approximate oracles. If $\alpha(x) = O(\log x)$, then the approximation ratio of the greedy algorithm is simply $O(\alpha(n) \cdot \log n)$. But, for example, if $\alpha(x) = x^{\varepsilon}$, then the lemma "saves" a factor of $\log n$ and shows that the approximation ratio is $\frac{1}{\varepsilon} \cdot n^{\varepsilon}$. So this settles the first question.

Lemma 5.1 also helps settle the second question. In fact, it proves that the greedy algorithm (with an exact oracle) is a bicriteria algorithm in the following sense.

Claim 5.1

If the greedy algorithm is stopped when $\beta \cdot n$ elements are covered, then the cost of the partial cover is bounded by $\ln\left(\frac{1}{1-\beta}\right) \cdot w(C^*)$.

The greedy algorithm surly does well with the first set it selects, but what can we say about the remaining selections? Claim 5.1 quantifies how well the greedy algorithm does as a function of the portion of the covered elements. For example, if $\beta = 1 - 1/e$, then the partial cover computed by the greedy algorithm weighs no more than $w(C^*)$. (We ignore here the knapsack-like issue of how to cover "exactly" $\beta \cdot n$ elements, and assume that, when we stopped the greedy algorithm, the partial cover covers $\beta \cdot n$ elements.) The lesson to be remembered here is that the greedy algorithm performs "reasonably well" as long as "few" elements have been covered.

The DST problem is a generalization of the SC problem. In fact, every SC instance can be represented as a DST instance over a layered directed graph with three vertex layers (see Figure 5.1). The top layer contains only a root, the middle layer contains a vertex for every set, and the bottom layer contains a vertex for every element. The weight of an edge from the root to a set is simply the weight of the set. The weight of all edges from sets to elements are zero. The best approximation algorithm for SC is the greedy algorithm. What form could a greedy algorithm have for the DST problem?

5.3 Directed Steiner Problems

In this section, we present three versions of DST problems. We present simple reductions that allow us to focus on only the last version.

Notation and Terminology

We denote the vertex set and edge set of a graph G by $V(G)$ and $E(G)$, respectively. An *arborescence* T rooted at r is a directed graph such that (i) the underlying graph of T is a tree (i.e., if edge directions are ignored in T, then T is a tree), and (ii) there is a directed path in T from the root r to every node in T. If an arborescence T is a subgraph of G, then we say that T *covers* (or *spans*) a subset of vertices X if $X \subseteq V(T)$. If edges have weights $w(e)$, then the weight of a subgraph G' is simply $\sum_{e \in E(G')} w(e)$. We denote by T_v the subgraph of T that is induced by all the vertices reachable from v (including v).

5.3.1 The Problems

The DST Problem

In the DST problem the input consists of a directed graph G, a set of terminals $X \subseteq V(G)$, positive edge weights $w(e)$, and a root $r \in V(G)$. An arborescence T rooted at r is a DST if it spans the set of terminals X. The goal in the DST problem is to find a minimum-weight DST.

The k-DST Problem

Following Ref. [3], we consider a version of the DST problem, called k-DST, in which only part of the terminals must be covered. In the k-DST problem, there is an additional parameter k, often called the *demand.* An arborescence T rooted at r is a k*-partial* DST (k-DST) if $|V(T) \cap X| \geq k$. The goal in the k-DST problem is to find a minimum-weight k-partial DST. We denote the weight of an optimal k-partial DST by $DS^*(G, X, k)$. (Formally, the root r should be a parameter, but we omit it to shorten the notation.) We encode DST instances as k-DST instances simply by setting $k = |X|$.

The ℓ-Shallow k-DST Problem

Following Ref. [2], we consider a version of the k-DST problem in which the length of the paths from the root to the terminals is bounded by a parameter ℓ. A rooted arborescence in which every node is at most ℓ edges away from the root is called an ℓ*-layered tree.* (Note that we count the number of layers of edges; the number of layers of nodes is $\ell + 1$.) In the ℓ-shallow k-DST problem, the goal is to compute a minimum k-DST among all ℓ-layered trees.

5.3.2 Reductions

Obviously, the k-DST problem is a generalization of the DST problem. Similarly, the ℓ-shallow k-DST problem is a generalization of the k-DST problem (i.e., simply set $\ell = |V| - 1$). The only nontrivial approximation algorithm we know is for the ℓ-shallow k-DST problem; this approximation algorithm is a recursive greedy algorithm. Since its running time is exponential in ℓ, we need to consider reductions that result with as small as possible values of ℓ.

For this purpose we consider two well-known transformations: transitive closure and layering. We now define each of these transformations.

Transitive Closure

The *transitive closure* of G is a directed graph $TC(G)$ over the same vertex set. For every $u, v \in V$, the pair (u, v) is an edge in $E(TC(G))$ if there is a directed path from u to v in G. The weight $w'(u, v)$ of an edge in $E(TC(G))$ is the minimum weight of a path in G from u to v.

The weight of an optimal k-DST is not affected by applying transitive closure namely,

$$DS^*(G, X, k) = DS^*(TC(G), X, k) \tag{5.3}$$

This means that replacing G by its transitive closure does not change the weight of an optimal k-DST. Hence, we may assume that G is transitively closed, i.e., $G = TC(G)$.

Layering

Let ℓ denote a positive integer. We reduce the directed graph G into an ℓ-layered directed acyclic graph LG_ℓ as follows (see Figure 5.2). The vertex set $V(LG_\ell)$ is simply $V(G) \times \{0, \ldots, \ell\}$. The jth layer in $V(LG_\ell)$ is the subset of vertices $V(G) \times \{j\}$. We refer to $V(G) \times \{0\}$ as the *bottom layer* and to $V(G) \times \{\ell\}$ as the *top layer.* The graph LG_ℓ is layered in the sense that $E(LG_\ell)$ contains only edges from the $V(G) \times \{j + 1\}$ to $V(G) \times \{j\}$, for $j < \ell$. The edge set $E(LG_\ell)$ contains two types of edges: *regular* edges and *parallel* edges. For every $(u, v) \in E(G)$ and every $j < \ell$, there is a regular

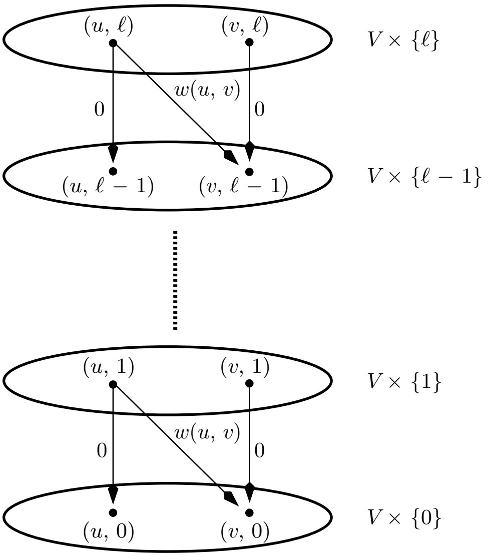


FIGURE 5.2 Layering of a directed graph G. Only parallel edges incident to images of $u, v \in V(G)$ and regular edges corresponding to $(u, v) \in E(G)$ are depicted.

edge $(u, j + 1) \rightarrow (v, j) \in E(LG_\ell)$. For every $u \in V$ and every $j < \ell$, there is a parallel edge $(u, j + 1) \rightarrow (u, j) \in E(LG_\ell)$. All parallel edges have zero weight. The weight of a regular edge is inherited from the original edge, namely, $w((u, j + 1) \rightarrow (v, j)) = w(u, v)$. The set of terminals X' in $V(LG_\ell)$ is simply $X \times \{0\}$, namely, images of terminals in the bottom layer. The root in LG_ℓ is the node (r, ℓ). The following observation shows that we can restrict our attention to layered graphs.

Observation 5.1

There is a weight- and terminal-preserving correspondence between ℓ-layered r-rooted trees in G and (r, ℓ)-rooted trees in LG_ℓ. In particular, $w(LT^*_\ell) = DS^*(LG_\ell, X', k)$, where LT^*_ℓ denotes a minimum-weight k-DST among all ℓ-layered trees.

Observation 5.1 implies that if we wish to approximate LT^*_ℓ, then we may apply layering and assume that the input graph is an ℓ-layered acyclic graph in which the root is in the top layer and all the terminals are in the bottom layer.

Limiting the Number of Layers

As we pointed out, the running time of the recursive greedy algorithm is exponential in the number of layers. It is therefore crucial to be able to bound the number of layers. The following lemma bounds the penalty incurred by limiting the number of layers in the Steiner tree. The proof of the lemma appears in Appendix A and uses notation introduced in Section 5.4. (A slightly stronger version appears in Ref. [8], with the ratio $2^{1-1/\ell} \cdot \ell \cdot k^{1/\ell}$.)

Lemma 5.2 (Zelikovsky [1], corrected in Helvig et al. [8])

If G is transitively closed, then $w(LT^*_\ell) \leq \frac{\ell}{2} \cdot k^{2/\ell} \cdot DS^*(G, X, k)$.

It follows that an α-approximate algorithm for an ℓ-shallow k-DST is also an $\alpha\beta$-approximation algorithm for k-DST, where $\beta = \frac{\ell}{2} \cdot k^{2/\ell}$. We now focus on the development of an approximation algorithm for the ℓ-shallow k-DST problem.

5.4 A Recursive Greedy Algorithm for ℓ-Shallow k-DST

This section presents a recursive greedy algorithm for the ℓ-shallow k-DST problem. Based on the layering transformation, we assume that the input graph is an ℓ-layered acyclic directed graph G. The set of terminals, denoted by X, is contained in the bottom layer. The root, denoted by r, belongs to the top layer.

5.4.1 Motivation

We now try to extend the greedy algorithm to the ℓ-shallow k-DST problem. Suppose we have a directed tree $T \subseteq G$ that is rooted at r. This tree only covers part of the terminals. Now we wish to augment T so that it covers more terminals. In other words, we are looking for an r-rooted augmenting tree T_{aug} to be added to the T. We follow the minimum density heuristic, and define the residual density of T_{aug} by

$$\rho_T(T_{aug}) \triangleq \frac{w(T_{aug})}{|(T_{aug} \cap X) \setminus (T \cap X)|}$$

All we need now is an algorithm that finds an augmenting tree with the minimum residual density. Unfortunately, this problem is by itself NP-hard. Consider the following reduction: Let G denote the two-layered DST instance mentioned above to represent an SC instance. Add a layer with a single node r' that is connected to the root r of G. The weight of the edge (r', r) should be large (say, n times the sum of the weights of the sets). It is easy to see that every minimum density subtree must span all the terminals. Hence, every minimum density subtree induces a minimum-weight SC, and finding a minimum density subtree in a three-layered graph is already NP-hard. We show in Section 5.4.3 that for two or less layers, one can find a minimum density augmenting tree in polynomial time.

We already showed that the greedy algorithm also works well with an approximate oracle. So we try to approximate a subtree with minimum residual density. The problem is how to do it? The answer is by applying a greedy algorithm recursively!

Consider an ℓ-layered directed graph and a root r. The algorithm finds a low-density ℓ-layered augmenting tree by accumulating low-density $(\ell - 1)$-layered augmenting trees that hang from the children of r. These trees are found by augmenting low-density trees that hang from grandchildren of r, and so on. We now formally describe the algorithm.

5.4.2 The Recursive Greedy Algorithm

Notation

We denote the number of terminals in a subgraph G' by $k(G')$ (i.e., $k(G') = |X \cap V(G')|$). Similarly, for a set of vertices U, $k(U) = |X \cap U|$. We denote the set of vertices reachable in G from u by *desc*(u). We denote the layer of a vertex u by *layer*(u) (e.g., if u is a terminal, then *layer*$(u) = 0$).

Description

A listing of the algorithm DS(u, k, X) appears as Algorithm 5.1. The stopping condition is when u belongs to the bottom layer or when the number of uncovered terminals reachable from u is less than the demand k (i.e., the instance is infeasible). In either case, the algorithm simply returns the root $\{r\}$.

The algorithm maintains a partial cover T that is initialized to the single vertex u. The augmenting tree T_{aug} is selected as the best tree found by the recursive calls to the children of u (together with the edge from u to its child). Note that the recursive calls are applied to all the children of u and all the possible demands k'. After T_{aug} is added to the partial solution, the terminals covered by T_{aug} are erased from the set of terminals so that the recursive calls will not attempt to cover terminals again. Once the demand is met, namely, k terminals are covered, the accumulated cover T is returned.

The algorithm is invoked with the root r, the demand k, and the set of terminals X. Note that if the instance is feasible (namely, at least k terminals are reachable from the root), then the algorithm never encounters infeasible subinstances during its execution.

Algorithm 5.1 DS(u, k, X)—A recursive greedy algorithm for the Directed Steiner Tree Problem. The graph is layered and all the vertices in the bottom layer are terminals. The set of terminals is denoted by X. We are searching for a tree rooted at u that covers k terminals.

```
stopping condition: if $layer(u) = 0$ or $k(desc(u)) < k$ then return $(\{u\})$.
initialize: $T \leftarrow \{u\}$; $X^{res} \leftarrow X$.
while $k(T) < k$ do
    recurse: for every $v \in children(u)$ and every $k' \leq \min\{k - k(T), |desc(v) \cap X^{res})|\}$
                    $T_{v,k'} \leftarrow \mathrm{DS}(v, k', X^{res})$.
    select: Let $T_{aug}$ be a lowest residual density tree among the trees $T_{v,k'} \cup \{(u, v)\}$, where
        $v \in children(u)$ and $k' \leq k - k(T)$.
    augment & update: $T \leftarrow T \cup T_{aug}$; $X^{res} \leftarrow X^{res} \setminus V(T_{aug})$.
end while
return $(T)$.
```

5.4.3 Analysis

Minimum Residual Density Subtree

Consider a partial solution T rooted at u accumulated by the algorithm. A tree T' rooted at u is a *candidate tree* for augmentation, if (i) every vertex $v \in V(T')$ in the bottom layer of G is in X^{res} (i.e., T' covers only new terminals) and (ii) $0 < k(T') \leq k - k(T)$ (i.e., T' does not cover more terminals than the residual demand). We denote by T'_u a tree with minimum residual density among all the candidate trees.

We leave the proof of the following lemma as an exercise.

Lemma 5.3

Assume that $w_i, k_i > 0$, *for every* $0 \leq i \leq n$. *Then,* $\min_i \frac{w_i}{k_i} \leq \frac{\sum_i w_i}{\sum_i k_i} \leq \max_i \frac{w_i}{k_i}$.

Corollary 5.1

If u is not a terminal, then we may assume that u has a single child in T'_u.

Proof

We show that we could pick a candidate tree with minimum residual density in which u has a single child. Suppose that u has more than one child in T'_u. To every edge $e_j = (u, v_j) \in E(T'_u)$ we match a subtree A_{e_j} of T'_u. The subtree A_{e_j} contains u, the edge (u, v_j), and the subtree of T'_u hanging from v_j. The subtrees $\{A_{e_j}\}_{e_j}$ form an edge-disjoint decomposition of T'_u. Let $w_j = w(A_{e_j})$ and $k_j = k(A_{e_j} \setminus T)$. Since u is not a terminal, the subtrees $\{A_{e_j}\}_{e_j}$ partition the terminals in $V(T'_u)$, and $k(T'_u) = \sum_j k_j$. Similarly, $w(T'_u) = \sum_j w_j$. By Lemma 5.3, it follows that one of the trees A_{e_j} has a residual density that is not greater than the residual density of T'_u. Use this minimum residual density subtree instead of T'_u, and the corollary follows. □

Density

Note that edge weights are nonnegative and already covered terminals do not help in reducing the residual density. Therefore, every augmenting tree T_{aug} covers only new terminals and does not contain terminals already covered by T. It follows that every terminal in T_{aug} belongs to X^{res} and, therefore, $k(T_{aug}) = |T_{aug} \cap X^{res}|$. We may assume that the same holds for T'_u; namely, T'_u does not contain already covered terminals. Therefore, where possible, we ignore the "context" T in the definition of the residual density and simply refer to *density*, i.e., the density of a tree T' is $\rho(T') = w(T')/|V(T') \cap X|$.

Notation and Terminology

A *directed star* is a one-layered rooted directed graph (i.e., there is a center out of which directed edges emanate to the leaves). We abbreviate and refer to a directed star simply as a star. A *flower* is a two-layered rooted graph in which the root has a single child.

Bounding the Density of Augmenting Trees

When *layer* $(u) = 1$, if u has least k terminal neighbors, then the algorithm returns a star centered at u. The number of edges emanating from r in the star equals k, and these k edges are the k lightest edges emanating from r to terminals. It is easy to see that in this case the algorithm returns an optimal k-DST.

The analysis of the algorithm is based on the following claim that bounds the ratio between the densities of the augmenting tree and T'_u.

Claim 5.2 **(Charikar et al. [3])**

If layer $(u) \geq 2$*, then, in every iteration of the while loop in an execution of* $DS(u, k)$*, the subtree* T_{aug} *satisfies*

$$\rho(T_{aug}) \leq (layer(u) - 1) \cdot \rho(T'_u)$$

Proof

The proof is by induction on *layer* (u). Suppose that *layer* $(u) = 2$. By Corollary 5.1, T'_u is a flower that consists of a star S_v centered at a neighbor v of u, the node u, and the edge (u, v). Moreover, S_v contains the $k(T'_u)$ closest terminals to v. When the algorithm computes T_{aug} it considers all stars centered at children v' of u consisting of the $k' \leq k - k(T)$ closest terminals to v'. In particular, it considers the star S_v together with the edge (u, v). Hence, $\rho(T_{aug}) \leq \rho(T'_u)$, as required.

We now prove the induction step for *layer* $(u) > 2$. Let $i =$ *layer* (u). The setting is as follows: During an execution of $DS(u, X)$, a partial cover T has been accumulated, and now an augmenting tree T_{aug} is computed. Our goal is to bound the density of T_{aug}.

By Corollary 5.1, u has a single child in T'_u. Denote this child by u'. Let $B_{u'}$ denote the subtree of T'_u that hangs from u' (i.e., $B_{u'} = T'_u \backslash \{u, (u, u')\}$). Let $k' = k(T'_u)$.

We now analyze the selection of T_{aug} while bearing in mind the existence of the "hidden candidate" T'_u that covers k' terminals. Consider the tree $T_{u',k'}$ computed by the recursive call DS(u', k', X^{res}). We would like to argue that $T_{u',k'}$ should be a good candidate. Unfortunately, that might not be true! However, recall that the greedy algorithm does "well" as long as "few" terminals are covered. So we wish to show that a "small prefix" of $T_{u',k'}$ is indeed a good candidate. We now formalize this intuition.

The tree $T_{u',k'}$ is also constructed by a sequence of augmenting trees, denoted by $\{A_j\}_j$. Namely, $T_{u',k'} = \bigcup_j A_j$. We identify the smallest index ℓ for which the union of augmentations $A_1 \cup \cdots \cup A_\ell$ covers at least $k'/(i - 1)$ terminals (recall that $i =$ *layer* (u)). Formally,

$$k\left(\bigcup_{j=1}^{\ell-1} A_j\right) < \frac{k'}{(i-1)} \leq k\left(\bigcup_{j=1}^{\ell} A_j\right)$$

Our goal is to prove the following two facts. Fact (1): Let $k'' = k(\bigcup_{j=1}^{\ell} A_j)$, then the candidate tree $T_{u',k''} =$ DS(u', k'', X^{res}) equals the prefix $\bigcup_{j=1}^{\ell} A_j$. Fact (2): The density of $T_{u',k''}$ is small, i.e., $\rho(T_{u',k''}) \leq (i - 1) \cdot \rho(B_{u'})$.

The first fact is a "simulation argument" since it claims that the union of the first ℓ augmentations computed in the course of the construction of $T_{u',k'}$ is actually one of the candidate trees computed by the algorithm. This simulation argument holds because, as long as the augmentations do not meet the demand, the same prefix of augmentations is computed. Note that k'' is the formalization of "few" terminals (compared to k'). Using $k'/(i-1)$ as an exact measure for a few terminals does not work because the simulation argument would fail.

The second fact states that the density of the candidate $T_{u',k''}$ is smaller than $(i - 1) \cdot \rho(B_{u'})$. Note that $B_{u'}$ and $A_1 \cup \cdots \cup A_{\ell-1}$ may share terminals (in fact, we would "like" the algorithm to "imitate"

$B_{u'}$ as much as possible). Hence, the residual density of $B_{u'}$ may increase as a result of adding the trees $A_1, \ldots, A_{\ell-1}$. However, since $k(A_1 \cup \cdots \cup A_{\ell-1}) < k'/(i-1)$, it follows that even after accumulating $A_1 \cup \cdots \cup A_{\ell-1}$, the residual density of $B_{u'}$ does not grow much. Formally, the residual density of $B_{u'}$ after accumulating $A_1 \cup \cdots \cup A_{\ell-1}$ is bounded as follows:

$$\begin{aligned}\rho_{(T \cup A_1 \cup \cdots \cup A_{\ell-1})}(B_{u'}) &= \frac{w(B_{u'})}{k' - k(A_1 \cup \cdots A_{\ell-1})}\\ &\leq \frac{w(B_{u'})}{k' \cdot (1 - \frac{1}{i-1})}\\ &= \left(\frac{i-1}{i-2}\right) \cdot \rho(B_{u'}) \end{aligned} \tag{5.4}$$

We now apply the induction hypothesis to the augmenting trees A_j (for $j \leq \ell$), and bound their residual densities by ($layer(u') - 1$) times the "deteriorated" density of $B_{u'}$. Formally, the induction hypothesis implies that when A_j is selected as an augmentation tree its density satisfies:

$$\begin{aligned}\rho(A_j) &\leq (i-2) \cdot \rho_{(T \cup A_1 \cdots \cup A_{j-1})}(B_{u'})\\ &\leq (i-1) \cdot \rho(B_{u'}) \quad \text{(by Eq. (5.4))}\end{aligned}$$

By Lemma 5.3, $\rho(\bigcup_{j=1}^{\ell} A_j) \leq \max_{j=1 \ldots \ell} \rho(A_j)$. Hence, $\rho(T_{u',k''}) \leq (i-1) \cdot \rho(B_{u'})$, and the second fact follows.

To complete the proof, we need to deal with the addition of the edge (u, u').

$$\begin{aligned}\rho(\{(u,u')\} \cup T_{u',k''}) &= \frac{w(u,u') + w(T_{u',k''})}{k''}\\ &\leq \frac{w(u,u')}{k'} \cdot (i-1) + \rho(T_{u',k''}) \quad \left(\text{since } k'' \geq \frac{k'}{i-1}\right)\\ &\leq (i-1) \cdot \rho(\{(u,u')\} \cup B_{u'}) \quad \text{(by fact [2])}\\ &= (i-1) \cdot \rho(T'_u)\end{aligned}$$

The claim follows since $\{(u, u')\} \cup T_{u',k''}$ is only one of the candidates considered for the augmenting tree T_{aug} and hence $\rho(T_{aug}) \leq \rho(\{(u,u')\} \cup T_{u',k''})$. □

Approximation Ratio

The approximation ratio follows immediately from Lemma 5.1.

Claim 5.3

Suppose that G is ℓ-layered. Then, the approximation ratio of Algorithm $DS(r, k, X)$ is $O(\ell \cdot \log k)$.

Running Time

For each augmenting tree, Algorithm $DS(u, k, X)$ invokes at most $n \cdot k$ recursive calls from children of u. Each augmentation tree covers at least one new terminal, so there are at most k augmenting trees. Hence, there are at most $n \cdot k^2$ recursive calls from the children of u. Let $time(\ell)$ denote the running time of $DS(u, k, X)$, where $\ell = layer(u)$. Then the following recurrence holds: $time(\ell) \leq (n \cdot k^2) \cdot time(\ell - 1)$. We conclude that the running time is $O(n^{\ell} \cdot k^{2\ell})$.

5.4.4 Discussion

Approximation of k-DST

The approximation algorithm is presented for ℓ-layered acyclic graphs. In Section 5.3.2, we presented a reduction from the k-DST problem to the ℓ-shallow k-DST problem. The reduction is based on layering and its outcome is an ℓ-layered acyclic graph. We obtain the following approximation result from this reduction.

Theorem 5.2 **(Charikar et al. [3])**

For every ℓ, there an $O(\ell^3 \cdot k^{2/\ell})$-approximation algorithm for the k-DST problem with running time $O(k^{2\ell} \cdot n^{\ell})$.

Proof

The preprocessing time is dominated by the running time of $DS(r, k, X)$ on the graph after it is transitively closed and layered into ℓ layers.

Let R^* denote a minimum residual density augmenting tree in the transitive closure of the graph (without the layering). Let T'_{k^*} denote a minimum residual subtree rooted at u in the layered graph among the candidate trees that cover $k(R^*)$ terminals. By Lemma 5.2, $w(T'_{k^*}) \leq \ell/2 \cdot k(R^*)^{\ell/2} \cdot w(R^*)$, and hence, $\rho(T'_{k^*}) \leq \ell/2 \cdot k(R^*)^{\ell/2} \cdot \rho(R^*)$. Since $\rho(T'_u) \leq \rho(T'_{k^*})$, by Claim 5.2 it follows that $\rho(T_{aug}) \leq (\ell - 1) \cdot \ell/2 \cdot k^{2/\ell} \cdot \rho(R^*)$.

We now apply Lemma 5.1. Note that $\int \frac{x^{2/\ell}}{x} dx = \frac{\ell}{2} \cdot x^{2/\ell}$. Hence, $w(T) = O(\ell^3 \cdot k^{2/\ell})$, where T is the tree returned by the algorithm, and the theorem follows. □

We conclude with the following result.

Corollary 5.2

For every constant $\varepsilon > 0$, there exists a polynomial-time $O(k^{1/\varepsilon})$-approximation algorithm for the k-DST problem. There exists a quasi-polynomial-time $O(\log^3 k)$-approximation algorithm for the k-DST problem.

Proof

Substitute $\ell = 2/\varepsilon$ and $\ell = \log k$ in Theorem 5.2. □

Preprocessing

Computing the transitive closure of the input graph is necessary for the correctness of the approximation ratio. Recall that Lemma 5.2 holds only if G is transitively closed.

Layering, on the other hand, is used to simplify the presentation. Namely, the algorithm can be described without layering (see Refs. [2,3]). The advantage of using layering is that it enables a unified presentation of the algorithm (i.e., there is no need to deal differently with one-layered trees). In addition, the layered graph is acyclic, so we need not consider multiple "visits" of the same node. Finally, for a given node u, we know from its layer what the recursion level is (i.e., the recursion level is $\ell - \mathit{layer}(u)$) and what the height of the tree we are looking for is (i.e., current height is $\mathit{layer}(u)$).

Suggestions for Improvements

One might try to reduce the running time by not repeating computations associated with the computations of candidate trees. For example, when computing the candidate $T_{v,k-k(T)}$ the algorithm computes a sequence of augmenting trees that is used to build also other candidates rooted at v that cover fewer terminals (we relied on this phenomenon in the simulation argument used in the proof of Claim 5.2). However, such improvements do not seem to reduce the asymptotic running time; namely, the running time would still be exponential in the number of layers and the basis would still be polynomial. We discuss other ways to reduce the running time in the next section.

Another suggestion to improve the algorithm is to zero the weight of edges when they are added to the partial cover T (see Ref. [1]). Unfortunately, we do not know how to take advantage of such a modification in the analysis and, therefore, keep the edge weights unchanged even after we pay for them.

5.5 Improving the Running Time

In this section, we consider a setting in which the recursive greedy algorithm can be modified to obtain a poly-logarithmic approximation ratio in polynomial time. The setting is with a problem called the GS problem, and only part of the modifications are applicable also to the k-DST problem. (Recall that the problem of finding a polynomial-time poly-logarithmic approximation algorithm for k-DST is still open.)

Motivation

We saw that the running time of the recursive greedy algorithm is $O((nk^2)^\ell)$, where k is the demand (i.e., number of terminals that needs to be covered), the degree of a vertex can be as high as $n-1$ (since transitive closure was applied), and ℓ is the bound on the number of layers we allow in the k-DST.

To obtain polynomial running times, we first modify the algorithm and preprocess the input so that its running time is $\log(n)^{O(\ell)}$. We then set $\ell = \log n / \log\log n$. Note that

$$(\log n)^{\frac{\log n}{\log\log n}} = n$$

Hence, a polynomial running time is obtained.

Four modifications are required to make this idea work:

1. Bound the number of layers—we already saw that the penalty incurred by limiting the number of layers can be bounded. In fact, according to Lemma 5.2, the penalty incurred by $\ell = \log n / \log\log n$ is poly-logarithmic (since $\ell \cdot k^{2/\ell} = (\log n)^{O(1)}$).
2. Degree reduction—we must reduce the maximum degree so that it is poly-logarithmic, otherwise too many recursive calls are invoked. Preprocessing of GS instances over trees achieves such a reduction in the degree.
3. Avoid small augmenting trees—we must reduce the number of iterations of the while loop. The number of iterations can be bounded by $(\log n)^c$ if we require that every augmenting tree must cover at least a poly-logarithmic fraction of the residual demand.
4. Geometric search—we must reduce the number of recursive calls. Hence, instead of considering all demands below the residual demand, we consider only demands that are powers of $(1+\varepsilon)$.

The GS Problem over Trees

We now present a setting where all four modifications can be implemented. In the GS problem over trees, the input consists of: (1) an undirected tree T rooted at r with nonnegative edge edges w(e), and (2) groups $g_i \subseteq V(T)$ of terminals. A subtree $T' \subseteq T$ rooted at r covers k groups if $V(T')$ intersects at least k groups. We refer to a subtree that covers k groups as a k-GS tree. The goal is to find a minimum-weight k-GS tree.

We denote the number of vertices by n and the number of groups by m. For simplicity, assume that every terminal is leaf of T and that every leaf of T is a terminal. In addition, we assume that the groups g_i are disjoint. Note that the assumption that the groups are disjoint implies that $\sum_{i=1}^{m} |g_i| \leq n$.

Bounding the Number of Layers

Lemma 5.2 applies also to GS instances over trees, provided that transitive closure is used. Before transitive closure is used, we direct the edges from the node closer to the root to the node farther away from the root. As mentioned above, limiting the number of layers to $\ell = \log n / \log\log n$ incurs a poly-logarithmic penalty.

However, there is a problem with bounding the number of layers according to Lemma 5.2. The problem is that we need to transitively close the tree. This implies that we lose the tree topology and end up with an directed acyclic graph instead. Unfortunately, we only know how to reduce the maximum degree of trees, not of directed acyclic graphs. Hence, we need to develop a different reduction that keeps the tree topology.

In Ref. [6], a height reduction for trees is presented. This reduction replaces T by an ℓ-layered tree T'. The penalty incurred by this reduction is $O(n^{c/\ell})$, where c is a constant. The details of this reduction appear in Ref. [6].

Reducing the Maximum Degree

We now sketch how to preprocess the tree T to obtain a tree $\nu(T)$ such that: (i) There is a weight preserving correspondence between k-GS trees in T and in $\nu(T)$. (ii) The maximum number of children of a vertex in $\nu(T)$ is bounded by an integer $\beta \geq 3$. (iii) The number of layers in $\nu(T)$ is bounded by the number of layers in T plus $\lfloor \log_{\beta/2} n \rfloor$. We set $\beta = \lceil \log n \rceil$, and obtain the required reduction.

We define a node $v \in V(T)$ to be *β-heavy* if the number of terminals that are descendents of v is at least n/β; otherwise v is *β-light*.

Given a tree T rooted at u and a parameter β, the tree $\nu(T)$ is constructed recursively as follows. If u is a leaf, then the algorithm returns u. Otherwise, the star induced by u and its children is locally transformed as follows. Let $v_1, v_2, \ldots, v_k$ denote the children of u.

1. Edges between u and β-heavy children v_i of u are not changed.
2. The β-light children of u are grouped arbitrarily into minimal bunches such that each bunch (except perhaps for the last) is β-heavy. Note that the number of leaves in each bunch (except perhaps for the last bunch) is in the half-closed interval $[n_u/\beta, 2n_u/\beta)$. For every bunch B, a new node b is created. An edge (u, b) is added as well as edges between b and the children of u in the bunch B. The edge weights are set as follows: (a) $w(u, b) \leftarrow 0$, and (b) $w(b, v_i) \leftarrow w(u, v_i)$.

After the local transformation, let $v'_1, v'_2, \ldots, v'_j$ be the new children of u. Some of these children are the original children and some are the new vertices introduced in the bunching. The tree $\nu(T)$ is obtained by recursively processing the subtrees $T_{v'_i}$, for $1 \leq i \leq j$, in essence replacing $T_{v'_i}$ by $\nu(T_{v'_i})$.

The maximum number of children after processing is at most β because the subtrees $\{T_{v'_i}\}_i$ partition the nodes of $V(T_u) - \{u\}$ and each tree except, perhaps one, is β-heavy. The recursion is applied to each subtree $T_{v'_i}$, and hence $\nu(T)$ will satisfies the degree requirement, as claimed. The weight preserving correspondence between k-GS trees in T and in $\nu(T)$ follows from the fact that the "shared" edges (u, b) that were created for bunching together β-light children of u have zero weight.

We now bound the height of $\nu(T)$. Consider a path p in $\nu(T)$ from the root r to a leaf v. All we need to show is that p contains at most $\log_{\beta/2} n$ new nodes (i.e., nodes corresponding to bunches of β-light vertices). However, the number of terminals hanging from a node along p decreases by a factor of at least $\beta/2$ every time we traverse such a new node, and the bound on the height of $\nu(T)$ follows.

The Modified Algorithm

We now present the modified recursive greedy algorithm for GS over trees. A listing of the modified recursive greedy algorithm appears as Algorithm 5.2.

Algorithm 5.2 Modified-GS(u, k, $\mathcal{G}$)—Modified recursive greedy algorithm for k-GS over trees.

```
stopping condition: if $u$ is a leaf then return $(\{u\})$.
Initialize: $cover \leftarrow \{u\}$ and $\mathcal{G}^{res} \leftarrow \mathcal{G}$.
while $k(cover) < k$ do
    recurse: for every $v \in$ children($u$) and
    for every $k'$ $\underline{\text{power of } (1+\lambda) \text{ in } [\gamma_r \cdot (k - k(cover)), k - k(cover)]}$

        $T_{v,k'} \leftarrow$ Modified-GS($v$, $k'$, $\mathcal{G}^{res}$).

    select: (pick the lowest density tree)

        $T_{aug} \leftarrow \text{MIN-DENSITY}\left\{T_{v,k'} \cup \{(u, v)\}\right\}$.

    augment & update: $cover \leftarrow cover \cup T_{aug}$; $\mathcal{G}^{res} \leftarrow \mathcal{G}^{res} \setminus \{g_i : T_{aug}$ intersects $g_i\}$.
    keep $k/h(T_u)$-cover: $\underline{\text{if first time } k(cover) \geq k/h(T_u) \text{ then } cover_h \leftarrow cover.}$
end while
return $\underline{\text{(lowest density tree} \in \{cover, cover_h\})}$.
```

The following notation is used in the algorithm. The input is a rooted undirected tree T that does not appear as a parameter of the input. Instead, a node u is given, and we consider the subtree of T that hangs from u. We denote this subtree by T_u. The partial cover accumulated by the algorithm is denoted by *cover*. The set of groups of terminals is denoted by $\mathcal{G}$. The set of groups of terminals not covered by *cover* is denoted by $\mathcal{G}^{res}$. The number of groups covered by *cover* is denoted by $k(cover)$. The height of a tree T_u is the maximum number of edges along a path from u to a leaf in T_u. We denote the height of T_u by $h(T_u)$.

Two parameters λ and γ_v appear in the algorithm. The parameter λ is set to equal $1/h(T)$. The parameter γ_v satisfies $1/\gamma_v = |children(v)| \cdot (1 + 1/\lambda) \cdot (1 + \lambda)$.

Lines that are significantly modified (compared to Algorithm 5.1) are underlined. In line 4, two modifications take place. First, the smallest demand is not one, but a poly-logarithmic fraction of the residual demand (under the assumption that the maximum degree and the height is poly-logarithmic). Second, only demands that are powers of $(1+\lambda)$ are considered. In line 7, the algorithm also stores the partial cover that first covers at least $1/h(T_u)$ of the initial demand k. This change is important for the simulation argument in the proof. Since the algorithm does not consider all the demands, we need to consider also the partial cover that the simulation argument points to. Finally, in line 9, we return the partial cover with the best density among *cover* and $cover_h$. Again, this selection is required for the simulation argument.

Note that modified-GS$(u, k, \mathcal{G})$ may return now a cover that covers less than k groups. If this happens in the topmost call, then one needs to iterate until a k-GS cover is accumulated.

The following claim is proved in Ref. [6]. It is analogous to Claim 5.2 and is proved by rewriting the proof while taking into account error terms that are caused by the modifications. Due to lack of space, we omit the proof.

Claim 5.4 **(Chekuri et al. [6])**

The density of every augmenting tree T_{aug} *satisfies*

$$\rho(T_{aug}) \le (1+\lambda)^{2h(T_u)} \cdot h(T_u) \cdot \rho(T'_u)$$

The following theorem is proved in Ref. [6]. The assumptions on the height and maximum degree are justified by the reduction discussed above.

Theorem 5.3

Algorithm modified-GS$(r, k, \mathcal{G})$ *is a poly-logarithmic approximation algorithm with polynomial running time for GS instances over trees with logarithmic maximum degree and* $O(\log n/\log\log n)$ *height.*

5.6 Discussion

In this chapter, we presented the recursive greedy algorithm and its analysis. The algorithm is designed for problems in which finding a minimum density augmentation of a partial solution is an NP-hard problem. The main advantages of the algorithm are its simplicity and the fact that it is a combinatorial algorithm. The analysis of the approximation ratio of the recursive greedy algorithm is nontrivial and succeeds in bounding the density of the augmentations.

The recursive greedy algorithm has not been highlighted as a general method, but rather as an algorithm for Steiner tree problems. We believe that it can be used to approximate other problems as well.

Open Problems

The quasi-polynomial-time $O(\log^3 k)$-approximation algorithm for DST raises the question of finding a polynomial-time algorithm with a poly-logarithmic approximation ratio for DST. In particular, the question is whether the running time of the recursive greedy algorithm for DST can be reduced by modifications or preprocessing.

Acknowledgments

I would like to thank Guy Kortsarz for introducing me to the recursive greedy algorithm and sharing his understanding of this algorithm with me. Guy also volunteered to read a draft. I thank Chandra Chekuri for many discussions related to this chapter. Lotem Kaplan listened and read drafts and helped me in

the search for simpler explanations. Thanks to the Max-Planck-Institut für Informatik where I had the opportunity to finish writing the chapter. Special thanks to Kurt Mehlhorn and his group for carefully listening to a talk about this chapter.

Appendix A

Proof of Lemma 5.2

We prove that given a k-DST T in a transitive closed directed graph G, there exists a k-DST T' such that: (i) T' is ℓ-layered and (ii) $w(T') \leq \frac{\ell}{2} \cdot k^{2/\ell} \cdot w(T)$. The proof uses the notation introduced in Section 5.4.

Notation

Consider a rooted tree T. The subtree of T that consists of the vertices hanging from v is denoted by T_v. Let $\alpha = k^{2/\ell}$. We say that a node $v \in V(T)$ is *α-heavy* if $k(T_v) \geq k(T)/\alpha$. A node v is *α-light* if $k(T_v) < k(T)/\alpha$. A node v is *minimally α-heavy* if v is α-heavy and all its children are α-light. A node v is *maximally α-light* if v is α-light and its parent is α-heavy.

Promotion

We now describe an operation called promotion of a node (and hence the subtree hanging from the node). Let G denote a directed graph that is transitively closed. Let T denote a rooted tree at r that is a subgraph of G. *Promotion* of $v \in V(T)$ is the construction of the rooted tree T' over the same vertex set with the edge set: $E(T') \triangleq E(T) \cup \{(r, v)\} \backslash \{(p(v), v)\}$. The promotion of v simply makes v a child of the root.

Height Reduction

The height reduction procedure is listed as Algorithm 5.3. The algorithm iteratively promotes minimally α-heavy nodes that are not children of the root, until every α-heavy node is a child of the root. The algorithm then proceeds with recursive calls for every maximally α-light node. There are two types of maximally α-light nodes: (1) children of promoted nodes, and (2) α-light children of the root (that have not been promoted).

Algorithm 5.3 HR(T, r, α)—A recursive height reduction algorithm. T is a tree rooted at r, and $\alpha > 1$ is a parameter.

```
stopping condition: if $V(T) = \{r\}$ then return $(\{r\})$.
$T' \leftarrow T$.
while $\exists v \in V(T') : v$ is minimally $\alpha$-heavy & $dist(r, v) > 1$ do
    $T' \leftarrow promote(T', v)$
end while
for all maximally $\alpha$-light nodes $v \in V(T')$ do
    $T' \leftarrow$ tree obtained from $T'$ after replacing $T'_v$ by HR$(T'_v, v, \alpha)$.
end for
return $(T')$.
```

The analysis of the algorithm is as follows. Let $h_\alpha(k(T))$ denote an upper bound on the height of the returned tree as a function of the number of terminals in T. The recursion is applied only to maximally α-light trees that are one or two edges away from the current root. It follows that $h_\alpha(k(T))$ satisfies the recurrence

$$h_\alpha(k') \leq h_\alpha(k'/\alpha) + 2$$

Therefore, $h_\alpha(k') \leq 2 \log_\alpha k'$.

Bounding the Weight

We now bound the weight of the tree T' returned by the height reduction algorithm. Note that every edge $e' \in E(T')$ corresponds to a path $path(e') \in T$. We say that an edge $e \in E(T)$ is *charged* by an edge $e' \in E(T')$ if $e \in path(e')$. If we can prove that every edge $e \in E(T)$ is charged at most β times, then $w(T') \leq \beta \cdot w(T)$.

We now prove that every edge $e \in E(T)$ is charged at most $\alpha \cdot \log_\alpha k(T)$ times. It suffices to show that every edge is charged at most α times in each level of the recursion. Since the number of terminals reduces by a factor of at least α in each level of the recursion, the recursion depth is bounded by $\log_\alpha k(T)$. Hence, the bound on the number of times that an edge is charged follows.

Consider an edge $e \in E(T)$ and one level of the recursion. During this level of the recursion, α-heavy nodes are promoted. The subtrees hanging from the promoted nodes are disjoint. Since every such subtree contains at least $k(T)/\alpha$ terminals, it follows that the number of promoted subtrees is at most α. Hence, the number of new edges $(r, v) \in E(T')$ from the root r to a promoted node v is at most α. Each such new edge charges every edge in $E(T)$ at most once, and hence every edge in $E(T)$ is charged at most α times in each recursive call. Note also that the recursive calls in the same level of the recursion are applied to disjoint subtrees. Hence, for every edge $e \in E(T)$, the recursive calls that charge e belong to a single path in the recursion tree.

We conclude that the recursion depth is bounded by $\log_\alpha k(T)$ and an edge is charged at most α times in each recursive call. Set $\ell = 2\log_\alpha k(T)$, and then $\alpha \log_\alpha k(T) = \frac{\ell}{2} \cdot k^{2/\ell}$. The lemma follows. □

References

[1] Zelikovsky, A., A series of approximation algorithms for the acyclic directed Steiner tree problem, *Algorithmica*, 18, 99, 1997.

[2] Kortsarz, G. and Peleg, D., Approximating the weight of shallow Steiner trees, *Discrete Appl. Math.*, 93, 265, 1999.

[3] Charikar, M., Chekuri, C., Cheung, T., Dai, Z., Goel, A., Guha, S., and Li, M., Approximation algorithms for directed Steiner problems, *J. Algorithms*, 33, 73, 1999.

[4] Reich, G. and Widmayer, P., Beyond Steiner's problem: a VLSI oriented generalization, *Proc. of Graph-Theoretic Concepts in Computer Science (WG-89)*, *Lecture Notes in Computer Science*, Vol. 411, Springer, Berlin, 1990, p. 196.

[5] Garg, N., Konjevod, G., and Ravi, R., A polylogarithmic approximation algorithm for the group Steiner tree problem, *J. Algorithms*, 37, 66, 2000. Preliminary version in *Proc. of SODA*, 1998, p. 253.

[6] Chekuri, C., Even, G., and Kortsarz, G., A greedy approximation algorithm for the group Steiner problem, *Discrete Appl. Math.*, 154(1), 15, 2006.

[7] Zosin, L. and Khuller, S., On directed Steiner trees, *Proc. of SODA*, 2002, p. 59.

[8] Helvig, C. H., Robins, G., and Zelikovsky, A., Improved approximation scheme for the group Steiner problem, *Networks*, 37(1), 8, 2001.

6

Linear Programming

Yuval Rabani
Technion—Israel Institute of Technology

6.1 Introduction

In this chapter we discuss the role of linear programming (LP) in the design and analysis of combinatorial approximation algorithms. Our emphasis is on NP-hard problems in combinatorial optimization. One aspect of their computational hardness is that such problems lack a good characterization of optimal solutions. Thus, approximating the optimum often involves finding a tight-as-possible bound on the optimal value that can be computed efficiently. LP is a powerful tool in deriving such bounds. The starting point is usually a formulation of the combinatorial optimization problem as an integer linear program. As a concrete example, consider the problem of VERTEX COVER. Given an undirected graph $G = (V, E)$ with nonnegative weights on the vertices $w : V \to \mathbb{N}$, we wish to find a minimum-weight set of vertices $V' \subset V$ such that for every $e \in E$, $e \cap V' \neq \emptyset$. This is a well-known NP-hard problem (see Ref. [1]), and here is a natural way to express it as an integer linear program. For every $i \in V$ assign an indicator variable $x_i \in \{0, 1\}$, indicating whether or not $i \in V'$. The constraints $e \cap V' \neq \emptyset$ can be expressed as $x_i + x_j \geq 1$, where $e = \{i, j\}$. The resulting program is

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{i \in V} w(i) x_i & & \\ \text{subject to} \quad & x_i + x_j \geq 1 & \forall \{i, j\} \in E \quad & (6.1) \\ & x_i \in \{0, 1\} & \forall i \in V \quad & (6.2) \end{aligned}$$

An ideal bound on the optimum can be derived by optimizing the same objective function over the convex hull of the integer solutions. As the vertices of the convex hull are integer solutions, this would yield an optimal solution. Unfortunately, the fact that VERTEX COVER is NP-hard implies that we are not aware of a concise representation of this linear program. In particular, the convex hull has an exponential number of facets, corresponding to an exponential number of linear constraints. A polynomial-time algorithm that, given a vector $x \in \mathbb{R}^V$, finds a violated constraint or verifies that x is in the convex hull (a so-called *separation oracle*) is unlikely to exist. However, we can compute a lower bound on the optimum by relaxing the integrality constraints (6.2). Thus we get the following *linear programming relaxation* for VERTEX COVER:

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{i \in V} w(i) x_i & & \\ \text{subject to} \quad & x_i + x_j \geq 1 & \forall \{i, j\} \in E \quad & \\ & x_i \geq 0 & \forall i \in V \quad & (6.3) \end{aligned}$$

Notice that in an optimal solution there is no reason to set any variable x_i to a value greater than 1, so we do not have to add explicitly the inequalities $x_i \leq 1, \forall i \in V$.

There are several ways in which such a bound can be used to derive an approximation algorithm. The most straightforward method is to solve the linear program, getting an assignment of potentially fractional values to the variables, then use these values to generate an integral solution. Such a procedure is called *rounding*. This chapter's focus is on rounding procedures. Other methods merely use the LP relaxation (or its dual) as a tool for analyzing the performance of an algorithm that does not explicitly solve the linear program. The performance of the approximation algorithm is compared with the solution to the linear program, rather than to the optimal solution of the NP-hard optimization problem. We will not discuss such methods (including the *primal-dual schema* and *dual fitting*) here. These methods are discussed in Chapters 2, 4, 13, 37, 39, 40, 71, and 82. Additional LP rounding applications are discussed in Chapters 7, 9, 11, 12, 70, and 80.

An important invariant of an LP relaxation is its *integrality ratio*, which is the worst-case ratio, over all possible inputs to the combinatorial optimization problem, between the linear program's optimal value and the optimization problem's optimal value. Unless the relaxation is used in conjunction with other techniques, the integrality ratio usually determines our expectation as to the best guarantee that can be achieved by an approximation algorithm relying on the relaxation.

In the following section, we review some of the methods used to derive approximation algorithms from LP relaxations, following the example of VERTEX COVER and many other examples.

6.2 Rounding

How tight is the lower bound $\min\left\{\sum_{i\in V} w(i)x_i : \text{(6.1) and (6.3)}\right\}$? Consider the clique K_n with unit weights on the vertices. Clearly, any $n-1$ vertices form a vertex cover, and if at least two vertices are excluded from the solution, then there will be at least one edge that is not covered. Thus, the cost of an optimal vertex cover is $n-1$. In contrast, assigning $x_i = \frac{1}{2}$ to all vertices i is a feasible solution to VC-LP, and its cost is $\frac{n}{2}$. So the integrality ratio of VC-LP is at least $2-\frac{2}{n}$. The following theorem proves that this is essentially tight.

Theorem 6.1 **(Nemhauser and Trotter [2])**

Every basic feasible solution to the system of linear inequalities consisting of (6.1) *and* (6.3) *is half-integral.*

Proof

Consider a feasible solution x that is not half-integral. We may assume that no entry of x exceeds 1, otherwise it is not a basic solution. Let $V^- = \{i \in V : 0 < x_i < \frac{1}{2}\}$ and let $V^+ = \{i \in V : \frac{1}{2} < x_i < 1\}$. At least one of these sets is not empty. Let $\epsilon > 0$ be a real number such that for every $i \in V^-$, $0 < x_i \pm \epsilon < \frac{1}{2}$, and for every $i \in V^+$, $\frac{1}{2} < x_i \pm \epsilon < 1$. Put

$$x_i^- = \begin{cases} x_i + \epsilon, & i \in V^- \\ x_i - \epsilon, & i \in V^+ \\ x_i, & i \in V\setminus(V^- \cup V^+) \end{cases}$$

and

$$x_i^+ = \begin{cases} x_i - \epsilon, & i \in V^- \\ x_i + \epsilon, & i \in V^+ \\ x_i, & i \in V\setminus(V^- \cup V^+) \end{cases}$$

Both x^- and x^+ are feasible solutions, $x^- \neq x^+$, and $x = \frac{1}{2}(x^- + x^+)$. Therefore, x cannot be a basic solution. □

Theorem 6.1 immediately leads to the following approximation algorithm, due to Hochbaum [3]. Solve the above linear program obtaining a basic optimal solution x^*. Set $V' = \{i \in V : x_i^* \geq \frac{1}{2}\}$. Clearly V' is a vertex cover. (In fact, we do not even need half-integrality. Every feasible solution x has the property that for every edge $\{i, j\} \in E$, either $x_i \geq \frac{1}{2}$ or $x_j \geq \frac{1}{2}$.) Also, by the choice of V' we have that

$$\sum_{i \in V'} w(i) \leq 2 \sum_{i \in V} w(i) x_i^*$$

Therefore, the above algorithm gives a 2 approximation to VERTEX COVER.

Half-integral relaxations that can be rounded easily are rare. In most cases where rounding works, it requires far greater effort and sophistication. Consider the problem of scheduling jobs on a set of unrelated machines so as to minimize the makespan, or R||C$_{\text{max}}$ in standard scheduling theory notation. The input to this problem consists of m machines and n jobs; job j is a sequence of $p_{1j}, p_{2j}, \ldots, p_{mj}$, where $p_{ij} \in \mathbb{N} \cup \{\infty\}$ denotes the processing time of job j on machine i. The goal is to find an assignment of jobs to machines $\varphi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, m\}$ that minimizes the makespan (i.e., the maximum load on a machine), which is

$$\max_i \sum_{j \in \varphi^{-1}(i)} p_{ij}$$

Clearly, this problem can be solved by the combination of binary search on the minimum makespan M, and a procedure to decide if a solution with makespan at most M exists and provide such a solution if it exists. An obvious formulation of the decision problem as an integer solution to a set of linear constraints uses indicator variables $x_{ij} \in \{0, 1\}$, where $x_{ij} = 1$ if and only if job j is assigned to machine i. Formally, the set of constraints is

$$\sum_i x_{ij} \geq 1 \quad \forall j \tag{6.4}$$

$$\sum_j p_{ij} x_{ij} \leq M \quad \forall i \tag{6.5}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \tag{6.6}$$

The first set of constraints (6.4) ensures that every job is assigned to a machine. The second set of constraints (6.5) ensures that the load on each machine is at most the target M. A 0–1 vector x that satisfies all of the above constraints corresponds to a solution with makespan at most M. As R||C$_{\text{max}}$ is NP-hard, finding a feasible solution x is NP-hard. We thus relax the integrality constraints (6.6), replacing them by the constraints

$$x_{ij} \geq 0 \quad \forall i, j \tag{6.7}$$

One last modification is necessary to make this relaxation useful. A fractional solution may assign a fraction of a job j to a machine i for which $p_{ij} > M$. This clearly cannot happen in an integer solution, but might happen in a fractional solution. We therefore eliminate from the inequalities all variables x_{ij} for which $p_{ij} > M$.

A 2-approximation algorithm for R||C$_{\text{max}}$ based on solving the resulting system of linear inequalities is a trivial consequence of the following theorem.

Theorem 6.2 (Lenstra et al. [4])

Any basic solution to the above system of linear inequalities can be rounded in polynomial time to give an assignment of jobs to machines such that the load on any machine does not exceed $2M$.

Proof

The number of constraints (6.4) and (6.5) is $n + m$, so a basic solution x has at most $n + m$ nonzero entries. Any job j that is assigned fractionally to two or more machines contributes at least two nonzero entries. Therefore, at most m jobs are assigned fractionally. To round the fractional solution, assign job j to machine i whenever $x_{ij} = 1$. Let the set of remaining jobs be J. As explained, $|J| \leq m$. Find a matching $\varphi : J \to \{1, 2, \ldots, m\}$, where a job $j \in J$ is matched to a machine $i = \varphi(j)$ such that $x_{ij} > 0$. (The proof that such a matching exists is rather involved. We do not include it here.) Assign the jobs in J according to the matching φ. The analysis of the performance guarantee is quite simple. Assume that a fractional basic solution x exists. The load due to the jobs not in J is at most the fractional load, which is at most M. A job $j \in J$ adds a load of at most M to $\varphi(j)$, because $x_{ij} > 0$ implies that $p_{ij} \le M$. Therefore, the total load on any machine is at most $2M$. □

Another useful tool is *filtering*, which is a technique to exclude some nonzero expensive entries in a fractional solution, leaving a "critical mass" that can be rounded with provable performance. This technique was first proposed by Lin and Vitter [5]. Here we demonstrate its use in getting a constant factor approximation algorithm for METRIC FACILITY LOCATION, following the work of Shmoys et al. [6]. For the sake of simplifying the presentation, we do not try to optimize the performance guarantee. In one simple version of METRIC FACILITY LOCATION we are given a finite metric space (X, d) and a cost function on the points $f : X \to \mathbb{N}$. The points represent both clients, each having a unit demand, and potential locations for facilities. The cost of constructing a facility at $i \in X$ is $f(i)$. Each client $j \in X$ is served by the closest facility i at cost $d(i, j)$. The goal is to minimize the total construction cost plus the total service cost. Here is one way to express the problem using mixed integer programming.

$$\begin{array}{lccr} \text{minimize} & \sum_{i\in X} f(i)x_i + \sum_{i,j\in X} d(i,j)y_{ij} & & \\ \text{subject to} & y_{ij} \le x_i & \forall i, j \in X & (6.8) \\ & \sum_{i\in X} y_{ij} \ge 1 & \forall j \in X & (6.9) \\ & x_i \in \{0,1\} & \forall i \in X & (6.10) \end{array}$$

The obvious LP relaxation replaces the integrality constraints (6.10) with

$$x_i \ge 0 \quad \forall i \in X \tag{6.11}$$

This relaxation can be used to derive a constant factor approximation algorithm for METRIC FACILITY LOCATION, as the following theorem states.

Theorem 6.3

There is a polynomial-time algorithm that computes, for every vectors x, y that satisfy the constraints (6.8), (6.9), *and* (6.11), *integral vectors x', y' satisfying the same constraints, such that*

$$\sum_{i\in X} f(i)x'_i + \sum_{i,j\in X} d(i,j)y'_{ij} \le 4\left(\sum_{i\in X} f(i)x_i + \sum_{i,j\in X} d(i,j)y_{ij}\right)$$

Proof

We may assume that for every $j \in X$, $\sum_{i\in X} y_{ij} = 1$, otherwise we can scale y_{*j} without violating the constraints and without increasing the cost of the solution. For every $j \in X$, let $\rho_j = \sum_{i\in X} d(i,j)y_{ij}$ be the expected service cost for client j under the distribution y_{*j}. By Markov's Inequality, at least $\frac{1}{4}$ of the mass of the distribution lies on potential facility locations i with $d(i, j) \le \frac{4}{3}\rho_j$. We would like to choose the cheapest of these facility locations to open a facility there and serve client j. However, the problem with this idea is that the sets of close facilities for different clients may overlap partially, thus we may charge the same probability mass several times. To overcome this difficulty, we consider a set of clients that have disjoint sets of close facilities. Sort the clients by nondecreasing order of ρ_j, then select a maximal sequence of clients J such that the balls $B(j, 4\rho_j/3) = \{i \in X : d(i, j) \le 4\rho_j/3\}$, for all $j \in J$, are all disjoint. In each ball, select the cheapest location i and set $x'_i = 1$ for all such i. Set $x'_i = 0$ for all other i. Finally, for every $j \in X$, set $y'_{ij} = 1$ for a location i with $x'_i = 1$ which is closest to j, and set $y'_{ij} = 0$ for all other locations i. We have that

$$\sum_{i\in X} f(i)x'_i \le 4\sum_{j\in J}\sum_{i\in B(j,\rho_j)} f(i)x_i \le 4\sum_{i\in X} f(i)x_i$$

Consider $j \in J$. Let i' be the location with $y'_{i'j} = 1$. The cost of serving j is

$$d(i', j) \leq \frac{4}{3}\rho_j = \frac{4}{3}\sum_{i \in X} d(i, j)y_{ij}$$

Finally, consider $j \notin J$. There exists $j' \in J$ such that $\rho_{j'} \leq \rho_j$ and $B(j', 4\rho_{j'}/3) \cap B(j, 4\rho_j/3) \neq \emptyset$. Let i' be the location with $y'_{i'j} = 1$. Then,

$$d(i', j) \leq \frac{4}{3}\rho_j + \frac{8}{3}\rho_{j'} \leq 4\rho_j = 4\sum_{i \in X} d(i, j)y_{ij}$$

This concludes the proof. □

We conclude this section with a brief description of Jain's iterative rounding method, which he used to derive a 2-approximation algorithm for the GENERALIZED STEINER NETWORK problem [7]. The input to this problem is an undirected graph $G = (V, E)$, edge costs $c : E \to \mathbb{N}$, and connectivity requirements $r : V \times V \to \mathbb{N}$. The goal is to find a subgraph $G' = (V, E')$ of minimum total edge cost $\sum_{e \in E'} c(e)$, such that for every $i, j \in V$ there are at least $r(i, j)$ edge-disjoint paths connecting i and j in G'. (Clearly, we may assume that r is symmetric, that is, $r(i, j) = r(j, i)$.) The basis for Jain's algorithm is the following formulation of GENERALIZED STEINER NETWORK. Let $f : 2^V \to \mathbb{Z}$ be defined by $f(S) = \max_{i \in S, j \notin S} r(i, j)$. Thus, $f(S)$ denotes the connectivity requirement across the cut $(S, V \backslash S)$. For every $e \in E$, let $x_e \in \{0, 1\}$ be an indicator variable that will be set to 1 if and only if e is included in the solution E'. Then, GENERALIZED STEINER NETWORK can be expressed as follows:

$$\begin{array}{lll} \text{minimize} & \sum_{e \in E} c(e)x_e & \\ \text{subject to} & \sum_{e : |e \cap S| = 1} x_e \geq f(S) & \forall S \subset V \qquad (6.12) \\ & x_e \in \{0, 1\} & \forall e \in E \qquad (6.13) \end{array}$$

The function f that is used here is *weakly supermodular*, which means that $f(V) = 0$, and for every $A, B \subseteq V$, either $f(A) + f(B) \leq f(A \backslash B) + f(B \backslash A)$ or $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$ (or both). In fact, Jain's approximation algorithm works for any weakly supermodular function f. It is based on the obvious LP relaxation of the above integer program, replacing the integrality constraints (6.13) by

$$0 \leq x_e \leq 1 \quad \forall e \in E \tag{6.14}$$

Note that the resulting linear program has an exponential number of constraints. However, often an efficient *separation oracle* can be designed, so the linear program can be solved in polynomial time. This is the case with GENERALIZED STEINER NETWORK; the separation oracle computes for every pair of nodes $i, j \in V$ a minimum cut in G with edge capacities x and checks if the cut capacity is at least $r(i, j)$. The approximation algorithm is based on the following theorem.

Theorem 6.4 (Jain [7])

If f is weakly supermodular, then for every basic solution x to the inequalities (6.12) *and* (6.14) *there exists $e \in E$ such that $x_e \geq \frac{1}{2}$.*

The proof of this theorem is quite complicated and is therefore not included here. Given an optimal basic solution x, we generate an integer solution x' as follows. For every $e \in E$ such that $x_e \geq \frac{1}{2}$, we set $x'_e = 1$. Let $E_1 = \left\{e \in E : x_e \geq \frac{1}{2}\right\}$. Then we recompute a basic fractional solution x^1 with the added condition that the edges in E_1 must be picked. To compute x^1, we solve the following linear program:

$$\begin{array}{lll} \text{minimize} & \sum_{e \in E \backslash E_1} c(e)x_e & \\ \text{subject to} & \sum_{e : |e \cap S| = 1} x_e \geq f(S) - |\{e \in E_1 : |e \cap S| = 1\}| & \forall S \subset V \qquad (6.15) \\ & 0 \leq x_e \leq 1 & \forall e \in E \backslash E_1 \qquad (6.16) \end{array}$$

It can be shown easily that if f is weakly supermodular then so is the function g given by $g(S) = f(S) - |\{e \in E_1 : |e \cap S| = 1\}|$, so Theorem 6.4 applies to x^1. We continue to round the solution iteratively and recompute a basic fractional solution to the remaining problem until no more edges need to be taken. This gives a 2 approximation for GENERALIZED STEINER NETWORK.

6.3 Randomized Rounding

Consider the problem of MAXIMUM COVERAGE. The input is a collection of subsets $S_1, S_2, \ldots, S_m$ over the base set $\{1, 2, \ldots, n\}$, and a positive integer $k \in \{1, 2, \ldots, m\}$. The goal is to find k subsets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ such that their union $\bigcup_{j=1}^{k} S_{i_j}$ has maximum cardinality. The following is a standard formulation of the problem:

$$\begin{array}{lll} \text{maximize} & \sum_{j=1}^{n} z_j & \\ \text{subject to} & z_j \le \min\left\{1, \sum_{i:\, j\in S_i} x_i\right\} & \forall j \in \{1, 2, \ldots, n\} \quad (6.17) \\ & \sum_{i=1}^{m} x_i = k & \quad (6.18) \\ & x_i \in \{0, 1\} & \forall i \in \{1, 2, \ldots, m\} \quad (6.19) \end{array}$$

Here the variable x_i is the indicator for including the set S_i in the solution. The variable z_j gets set to 1 if and only if j is covered by the sets taken in the solution. In the obvious LP relaxation, these variables are set to values in the interval $[0, 1]$. One interpretation of the relaxation is that now x_i stands for the probability of including S_i in the solution and z_j for the probability of covering j. However, we have no guarantee that a sample space with the desired probabilities exists. Nevertheless, this interpretation proves to be fruitful. Consider the following probabilistic algorithm. Pick k sets at random by sampling k times independently from the distribution Pr given by $\Pr[S_i] = \frac{x_i}{k}$. (Note that $\sum_i \Pr[S_i] = 1$, so this is indeed a distribution over the sets.) Let E_j denote the event that j is covered by the random choice of k sets. We have that

$$\begin{aligned} \Pr[E_j] &= 1 - \Pr[\bar{E}_j] \\ &= 1 - \left(1 - \frac{1}{k} \sum_{i:\, j\in S_i} x_i\right)^k \\ &\ge 1 - \left(1 - \frac{z_j}{k}\right)^k \\ &\ge \frac{e-1}{e} z_j \end{aligned}$$

This implies an approximation guarantee of $\frac{e-1}{e}$ as the expected number of elements covered is $\frac{e-1}{e}\sum_j z_j$, and $\sum_j z_j$ is an upper bound on the optimal value.

Often, bounds on large deviations are useful in the context of randomized rounding. Let $X_1, X_2, X_3, \ldots, X_n$ be independent indicator random variables with $\Pr[X_i = 1] = p_i$. Let $X = \sum_{i=1}^{n} X_i$. By the linearity of expectation, $E[X] = \sum_{i=1}^{n} p_i$. The following Chernoff-like bound is attributed to Spencer (see Ref. [8]).

Lemma 6.1

For every $\epsilon > 0$,

$$\Pr\left[X > (1+\epsilon)E[X]\right] < \left(\frac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}\right)^{E[X]}$$

For an application, consider Raghavan and Thompson's [9] problem of integral multicommodity flow CONGESTION MINIMIZATION that we present here in its simplest unit capacities version. Given a (directed) graph $G = (V, E)$ and a list of source–destination pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ (also called *commodities)*, find, for every $i \in \{1, 2, \ldots, k\}$, a path p_i in G from s_i to t_i, minimizing the maximum number of paths that cross a single arc. Consider the following equivalent integer programming formulation. For $i \in \{1, 2, \ldots, k\}$, let P_i denote the set of all simple paths in G from s_i to t_i. (Note that $|P_i|$ might be exponentially large in the input size. We will deal with this issue later.)

$$\begin{array}{llll}
\text{minimize} & u & & \\
\text{subject to} & \sum_{p \in P_i} f_p^i = 1 & \forall i \in \{1, 2, \ldots, k\} & (6.20) \\
& \sum_{i=1}^{k} \sum_{p \in P_i : p \ni e} f_p^i \le u & \forall e \in E & (6.21) \\
& f_p^i \in \{0, 1\} & \forall i \in \{1, 2, \ldots, k\}, \ \forall p \in P_i & (6.22)
\end{array}$$

If we relax the integrality constraints (6.22), we get a linear program whose solution assigns, for every commodity $i \in \{1, 2, \ldots, k\}$, a probability distribution f^i over P_i. (In other words, we get a fractional multicommodity flow solution.) As mentioned, the number of variables in this linear program is exponential in the input size. It is nevertheless possible to solve it in time polynomial in the input size using a separation oracle for the dual. A more standard approach is to replace this linear program with a polynomial-size linear program that has arc flow variables and flow conservation constraints. The computed flows can be decomposed into a polynomial number of flow paths (see, e.g., the book [10]). Either way, we will get distributions f^i that have polynomial-size support. Note that $\max\{1, u\}$ is a lower bound for MINIMUM CONGESTION, as the optimal value is at least 1. We will use this bound in analyzing the following approximation algorithm.

The algorithm applies randomized rounding. For every commodity $i \in \{1, 2, \ldots, k\}$, we choose at random a single path p according to the distribution f^i (i.e., the probability of choosing p is f_p^i). Consider an arc $e \in E$. Clearly, the probability that the path for a commodity i uses e is precisely $q_i = \sum_{p \in P_i : p \ni e} f_p^i$. Let X_i be an indicator random variable that is set to 1 if and only if commodity i uses e. Thus, the load on e is given by $X = \sum_{i=1}^{k} X_i$. We have that

$$\mathrm{E}[X] = \sum_{i=1}^{k} q_i = \sum_{i=1}^{k} \sum_{p \in P_i : p \ni e} f_p^i$$

which is exactly the load on e in the linear program. Unfortunately, if there is more than one arc in the graph we cannot guarantee that all the arcs *simultaneously* will not be loaded more than the expectation. However, by Lemma 6.1, for a constant $c > 0$,

$$\Pr\left[X > c \frac{\log |E|}{\log \log |E|} \max\{1, \mathrm{E}[X]\}\right] < \frac{1}{2|E|}$$

Therefore, with probability at least $\frac{1}{2}$, every arc carries at most $c \frac{\log |E|}{\log \log |E|} \max\{1, u\}$ paths. In fact, if u is large, then the approximation guarantee improves. For example, if $u = \log |E|$, we can apply Lemma 6.1 with a constant ϵ to get a constant factor approximation. As u grows further, the approximation guarantee approaches 1.

6.4 Metric Spaces

Some problems in combinatorial optimization, most notably problems involving cuts in undirected graphs, can be interpreted naturally as optimization over a class of metric spaces. For example, consider the MINIMUM MULTICUT problem, introduced by Klein et al. [11]. The input to this problem is a graph $G = (V, E)$ with nonnegative edge capacities $c : E \to \mathbb{N}$, a positive integer k, and a set of k pairs of nodes $T = \{\{s_1, t_1\}, \{s_2, t_2\}, \ldots, \{s_k, t_k\}\}$ called *terminal pairs*. The goal is to find a set of edges $F \subset E$ of minimum total capacity $c(F) = \sum_{e \in F} c(e)$ whose removal disconnects every pair of terminals in T. This problem can be formalized as the following integer program. Let P denote the set of paths in G connecting

s_i and t_i, for some $i \in \{1, 2, \ldots, k\}$.

$$\begin{aligned} &\text{minimize} && \textstyle\sum_{e\in E} c(e)x_e && \\ &\text{subject to} && \textstyle\sum_{e\in p} x_e \ge 1 && \forall p \in P \qquad (6.23) \\ & && x_e \in \{0, 1\} && \forall e \in E \qquad (6.24) \end{aligned}$$

As usual, an LP relaxation is derived by replacing the integrality constraints (6.24) with

$$x_e \ge 0 \quad \forall e \in E \tag{6.25}$$

Note that the resulting linear program may have an exponential number of constraints. However, this can be dealt with using the same methods explained in the discussion on CONGESTION MINIMIZATION in the previous section. Let x be any feasible solution to the linear program. One way to interpret the solution is the following. Define a semi-metric[1] d on V by setting $d(u, v)$ to be the shortest path between u and v under edge weights given by x. Then the constraints (6.23) put $d(s_i, t_i) \ge 1$ for all $i \in \{1, 2, \ldots, k\}$. In fact, every such metric d corresponds to a feasible solution x by setting, for every $e = (u, v) \in E$, $x_e = d(u, v)$. Our hope is to find a way to "round" the semi-metric d to a multicut without increasing the objective function too much. Note that a multicut F corresponds to a semi-metric δ on V that satisfies the constraints (6.23) and also has $\delta(u, v) \in \{0, 1\}$ for every $u, v \in V$. More specifically, $\delta(u, v) = 0$ if and only if u and v are in the same connected component after removing the edges in F.

Indeed, Garg et al. [12] analyzed such a rounding procedure which is based on earlier work of Leighton and Rao [13] on SPARSEST CUT, a problem that will be discussed below.

Theorem 6.5 (Garg et al. [12])

There is a polynomial-time algorithm that, given input $x \in \mathbb{R}^E$ that satisfies the constraints (6.23) *and* (6.25), *finds a multicut F such that $c(F) = O(\sum_{e\in E} c(e)x_e \log k)$.*

Proof

Let d be the semi-metric on V that is derived from x. Let $w \in V$ be a terminal (i.e., $v = s_i$ or $v = t_i$ for some $i \in \{1, 2, \ldots, k\}$). For $\rho \in [0, \infty)$, let E_ρ denote the set of edges $\{u, v\}$ such that $d(u, w) \le \rho$ and $d(v, w) > \rho$. Consider the function $f' : [0, \infty) \to \mathbb{N}$ that is given by $f'(\rho) = \sum_{e\in E_\rho} c(e)$. Let $f(\rho) = \frac{1}{k}\sum_{e\in E} c(e)x_e + \int_0^\rho f'(\xi)\, d\xi$, so $f'(\rho) = \frac{df(\rho)}{d\rho}$. Note that for every $\rho \in [0, \infty)$, $f(\rho) \le 2\sum_{e\in E} c(e)x_e$. Now,

$$\int_0^{1/3} \frac{f'(\rho)}{f(\rho)}\, d\rho = \ln\left(\frac{f(1/3)}{f(0)}\right) \le \ln k$$

Therefore, there exists $\rho \in [0, 1/3]$ such that $f'(\rho) \le 3f(\rho)\ln k$. (Such ρ is easily found in polynomial time.) Note that as $\rho \le \frac{1}{3}$, it is impossible that for any $i \in \{1, 2, \ldots, k\}$ both $d(w, s_i) \le \rho$ and $d(w, t_i) \le \rho$.

The multicut F is generated inductively as follows. Pick a terminal $w = w_1$. Find $\rho = \rho_1$ as explained above, and eliminate from G the set $\{v \in V : d(v, w_1) \le \rho_1\}$. Let $G^1 = (V^1, E^1)$ denote the remaining graph. Suppose that $w_1, w_2, \ldots, w_t$ have been picked already. Pick a terminal $w = w_{t+1}$ in G^t (i.e., $d(w_{t+1}, w_s) > \rho_s$ for all $s \in \{1, 2, \ldots, t\}$). If there is no such terminal, then output $F = \cup_{s=1}^t (E_{\rho_s} \cap E^s)$. Otherwise, find $\rho = \rho_{t+1}$ in G^t as explained above, and eliminate from G^t the set $\{v \in V^t : d(v, w_{t+1}) \le \rho_{t+1}\}$ to create G^{t+1}. Note that t never exceeds k.

[1] A semi-metric d on a set V is a function $d : V \times V \to \mathbb{R}$ that satisfies the following conditions: (i) $d(v, v) = 0$, for every $v \in V$; (ii) $\delta(u, v) = \delta(v, u)$, for every $u, v \in V$; and (iii) $\delta(u, v) + \delta(v, w) \ge \delta(u, w)$, for every $u, v, w \in V$.

By the above discussion, F is a multicut. For $\rho \in [0, \rho_s]$, let $f'_s(\rho) = \sum_{e \in E_{\rho_s} \cap E^s} c(e)$. As

$$\sum_{s=1}^{t} \left(\frac{1}{k} \sum_{e \in E} c(e)x_e + \int_0^{\rho_s} f'_s(\xi)\, d\xi \right) \leq 2 \sum_{e \in E} c(e)x_e$$

the $O(\log\, k)$ bound on the approximation guarantee follows. □

Consider the problem of SPARSEST CUT. Given an undirected graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$ and a demand function $h : V \times V \to \mathbb{N}$, the goal is to find a cut $(S, \bar{S})$ that minimizes the cut ratio

$$\frac{\sum_{e:\ |S \cap e| = 1} c(e)}{\sum_{u \in S \wedge v \in \bar{S}} h(u, v)}$$

Leighton and Rao [13] gave an $O(\log |V|)$ approximation algorithm for the case that h is uniform (i.e., $h(u, v) = 1$ for every pair of nodes $u, v \in V$), using the "region growing" technique discussed above in the context of MINIMUM MULTICUT. We now discuss the general case.

A cut $(S, \bar{S})$ in G partitions the node set V into two nonempty parts S and $\bar{S}$. We can associate with $(S, \bar{S})$ a *cut semi-metric* δ_S on V. The semi-metric δ_S is defined by $\delta_S(x, y) = 1$ if $x \neq y$ and $|\{x, y\} \cap S| = 1$; otherwise $\delta_S(x, y) = 0$. The cone of linear combinations of cut semi-metrics on V with nonnegative coefficients is precisely the cone of $|V|$-point subsets of L_1. Useful polytopes can be derived from this cone by adding linear constraints that normalize the maximum or average distance. In particular, SPARSEST CUT can be formalized as follows:

$$\begin{array}{lll} \text{minimize} & \sum_{\{u,v\}=e \in E} c(e) \sum_{\emptyset \neq S \subseteq V} \delta_S(u, v)\lambda_S & \\ \text{subject to} & \sum_{u,v \in V} h(u, v) \sum_{\emptyset \neq S \subseteq V} \delta_S(u, v)\lambda_S = 1 & \\ & \lambda_S \geq 0 & \forall \emptyset \neq S \subseteq V \end{array}$$

This is a linear program with an exponential number of variables. (Note that the solution might be a convex combination of optimal cuts.) In view of the NP-hardness of SPARSEST CUT, it is unlikely that this LP can be solved in time polynomial in the size of G. A polynomial-time solvable relaxation can be derived by extending the optimization over all semi-metrics, not just nonnegative linear combinations of cut semi-metrics. This gives the following LP relaxation for SPARSEST CUT:

$$\begin{array}{ll} \text{minimize} & \sum_{\{u,v\}=e \in E} c(e)d(u, v) \\ \text{subject to} & \sum_{u,v \in V} h(u, v)d(u, v) = 1 \\ & d \text{ is a semi-metric on } V \end{array} \tag{6.26}$$

The following lemma is crucial.

Lemma 6.2 (Bourgain [14])

There is a constant $\kappa > 0$ such that the following holds. Let d be a semi-metric on a finite set of points X. Then, there exists $n \in \mathbb{N}$ and a mapping $\varphi : X \to \mathbb{R}^n$ such that for every $x, y \in X$,

$$\frac{1}{\kappa \log |X|} d(x, y) \leq \|\varphi(x) - \varphi(y)\|_1 \leq d(x, y)$$

Let supp(h) denote the support of the demand function h, i.e., the set of pairs $u, v \in V$ such that $h(u, v) > 0$.

Theorem 6.6 (Aumann and Rabani [15]; Linial et al. [16])

There exists a constant $\kappa > 0$ such that the following holds. Let d be a semi-metric on V that satisfies the constraint (6.26). Then, one can find in polynomial time a cut $(S, \bar{S})$ in G such that

$$\sum_{\{u,v\}=e \in E} c(e)d(u, v) \leq \frac{\sum_{e:\ |S \cap e| = 1} c(e)}{\sum_{u \in S \wedge v \in \bar{S}} h(u, v)} \leq \kappa \cdot \sum_{\{u,v\}=e \in E} c(e)d(u, v) \log |\text{supp}(h)|$$

Proof

Use a modification of Bourgain's Lemma 6.2 to map d and L_1 semi-metric such that the distances between pairs of points in $\mathrm{supp}(h)$ do not shrink by more than a factor of $O(\log|\mathrm{supp}(h)|)$ and no distance expands. Then use the fact that any L_1 semi-metric can be expressed as a nonnegative linear combination of cut semi-metrics to find a cut with a good ratio. □

We note that the bounds in Theorems 6.5 and 6.6 asymptotically match the known integrality gaps. The bad examples are constructed using bounded degree expander graphs. See Refs. [12,13,15,16] for more details.

We conclude this section and the chapter with a discussion of spreading metrics, a class of relaxations introduced by Even et al. [17]. We will demonstrate the technique with the problem of MINIMUM LINEAR ARRANGEMENT. Given an undirected graph $G = (V, E)$, the goal is to find a bijection $\varphi: V \to \{1, 2, \ldots, |V|\}$ that minimizes $\sum_{\{u,v\}\in E} |\varphi(u) - \varphi(v)|$. Note that one property of any such bijection is that every subset of nodes $U \subseteq V$ must be "well spread" in the sense that for every $u \in U$, $\sum_{v\in U} |\varphi(u) - \varphi(v)| \geq \frac{1}{4}(|U|^2 - 1)$. This is precisely the property that the spreading metric relaxation for MINIMUM LINEAR ARRANGEMENT exploits. Rather than optimizing over all bijections (which is NP-hard), we optimize over all metrics that satisfy the spreading constraints. Formally, we solve the following LP relaxation:

$$\begin{array}{lll} \text{minimize} & \sum_{\{u,v\}\in E} d(u, v) & \\ \text{subject to} & \sum_{v\in U} d(u, v) \geq \frac{1}{4}(|U|^2 - 1) & \forall U \subseteq V, \quad \forall u \in U \\ & d \text{ is a metric on } V & \end{array} \tag{6.27}$$

Note that the number of constraints is exponential in the size of the input graph G. However, given a metric d that does not satisfy all the constraints (6.27), it is easy to find a violated constraint in polynomial time by examining all the polynomially many combinatorially distinct balls in d. Thus, the relaxation can be solved in polynomial time.

Theorem 6.7 (Rao and Richa [18])

Given a metric d on V that satisfies all the constraints (6.27), *one can find in polynomial time a bijection $\varphi: V \to \{1, 2, \ldots, |V|\}$ such that*

$$\sum_{\{u,v\}\in E} |\varphi(u) - \varphi(v)| \leq O(\log|V|) \cdot \sum_{\{u,v\}\in E} d(u, v)$$

Proof

We describe the "rounding" algorithm. If G is not connected, we can deal with each connected component separately, so we may assume without loss of generality that G is connected. Consider a node $s \in V$. Define levels with respect to s that are indexed by $i \in \mathbb{N}$. We say that an edge $\{u, v\} \in E$ is at level i if and only if $d(s, u) \leq i$ and $d(s, v) > i$. The weight w_i of level i is the total number of edges at level i. We say that level i has label k if and only if $2^k < w_i \leq 2^{k+1}$. Note that due to constraints (6.27), there are at least $\frac{1}{4}|V|$ levels with strictly positive weight. Thus, putting $D = \sum_{\{u,v\}\in E} d(u, v)$, there is a label k such that at least $\frac{1}{4\log D}|V|$ levels are labeled with k. Let these levels be $i_1, i_2, \ldots, i_m$. Let H_0 denote the subgraph induced by the nodes $v \in V$ such that $d(s, v) \leq i_1$. For $j = 1, 2, \ldots, m - 1$, let H_j denote the subgraph induced by the nodes $v \in V$ such that $i_j < d(s, v) \leq i_{j+1}$. Finally, let H_m denote the subgraph induced by the nodes $v \in V$ such that $i_m < d(s, v)$. Recursively apply the above procedure to each of the subgraphs H_j, $j \in \{0, 1, 2, \ldots, m\}$. The output linear arrangement is composed of the concatenation of the linear arrangements for these subgraphs.

The analysis of the performance guarantee follows by devising a charging scheme, stating the charged cost as a recurrence relation and bounding the recurrence solution. The analysis is rather technical and therefore it is excluded here. □

References

[1] Garey, M. R. and Johnson, D. S., *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.

[2] Nemhauser, G. L. and Trotter, L. E., Vertex packings: structural properties and algorithms, *Math. Program.*, 8, 232, 1975.

[3] Hochbaum, D. S., Approximation algorithms for the weighted set covering and node cover problems, *SIAM J. Comput.*, 11(3), 555, 1982.

[4] Lenstra, J. K., Shmoys, D. B., and Tardos, É., Approximation algorithms for scheduling unrelated parallel machines, *Math. Program.*, 46, 259, 1990.

[5] Lin, J.-H. and Vitter, J. S., ϵ-Approximations with minimum packing constraint violation, *Proc. of STOC,* 1992, p. 771.

[6] Shmoys, D. B., Tardos, É., and Aardal, K. I., Approximation algorithms for facility location problems, *Proc. of STOC,* 1997, p. 265.

[7] Jain, K., Factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1), 39, 2001.

[8] Raghavan, P., Probabilistic construction of deterministic algorithms: approximating packing integer programs, *JCSS*, 37(2), 130, 1988.

[9] Raghavan, P. and Thompson, C. D., Randomized rounding, *Combinatorica*, 7, 365, 1987.

[10] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, New York, 1993.

[11] Klein, P., Agrawal, A., Ravi, R., and Rao, S., Approximation through multicommodity flow, *Proc. of FOCS,* 1990, p. 726.

[12] Garg, N., Vazirani, V. V., and Yannakakis, M., Approximate max-flow min-(multi)cut theorems and their applications, *SIAM J. Comput.*, 25(2), 235, 1996.

[13] Leighton, F. T. and Rao, S., Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms, *JACM*, 46(6), 787, 1999.

[14] Bourgain, J., On Lipschitz embedding of finite metric spaces in Hilbert space, *Israel J. Math.*, 52, 46, 1985.

[15] Aumann, Y. and Rabani, Y., An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm, *SIAM J. Comput.*, 27(1), 291, 1998.

[16] Linial, N., London, E., and Rabinovich, Y., The geometry of graphs and some of its algorithmic applications, *Combinatorica*, 15(2), 215, 1995.

[17] Even, G., Naor, J., Rao, S., and Schieber, B., Divide-and-conquer approximation algorithms via spreading metrics, *JACM*, 47(4), 585, 2000.

[18] Rao, S. and Richa, A. W., New approximation techniques for some linear ordering problems, *SIAM J. Comput.*, 34(2), 388, 2004.

7

LP Rounding and Extensions

Daya Ram Gaur
University of Lethbridge

Ramesh Krishnamurti
Simon Fraser University

7.1 Introduction

Many combinatorial optimization problems can be cast as integer linear programming problems. A linear programming relaxation of an integer program provides a natural lower bound (in case of minimization problems) on the value of the optimal integral solution. An optimal solution to the linear programming relaxation may not necessarily be integral. If there exists a procedure to obtain an integral solution "close" to the fractional solution then we have an approximation algorithm. This process of obtaining the integral solution from the fractional one is referred to as "rounding." Our goal is to present an ensemble of rounding techniques (which is by no means complete) that have enjoyed some success. On occasion, for detailed correctness of proofs, we refer the reader to the original paper.

Rounding techniques can be broadly divided into two categories: those that round variables nondeterministically (also called as randomized rounding), and those that round variables deterministically. The randomized rounding techniques presented typically yield solutions whose expected value is bounded. At times, the rounding steps can be made deterministic (derandomized) by using the method of conditional expectation due to Erdős and Selfridge [1]. We refer the reader to Alon and Spencer ([2], Chapter 15) for the method of conditional expectation. Both randomized as well as deterministic rounding can be further classified into techniques that round the variables independently, and those that round the variables in groups (dependently). Our presentation is along similar lines; in Section 7.2 we discuss nondeterministic rounding techniques due to Raghavan and Thompson [3], Goemans and Williamson [4], Bertsimas et al. [5], Goemans and Williamson [6], and Arora et al. [7]. We discuss deterministic rounding techniques due to Lin and Vitter [8], Jain [9], Ageev and Sviridenko [10], and Gaur et al. [11] in Section 7.3. Finally we conclude with a discussion. For other applications of rounding we refer the reader to the books by Hochbaum [12] and Vazirani [13].

Next, we define the performance ratio of an approximation algorithm. Associated with every instance $\mathcal{I}$ of an NP-optimization problem $\mathcal{P}$ is a nonempty set of feasible solutions $\mathcal{S}$. To each solution $S \in \mathcal{S}$, we assign a number called its value. For a minimization (maximization) problem, the goal is to determine the solution with the minimum (maximum) value. The solution with the minimum (maximum) value is denoted as $OPT(\mathcal{I})$ or simply as OPT when there is no ambiguity. Let A be an algorithm whose running time is bounded by a polynomial in the length of the input. $ALG(\mathcal{I})$ (or simply ALG) denotes the value of the solution returned by algorithm A on problem $\mathcal{P}$. For a minimization (maximization) problem, the performance ratio of A is defined as $\alpha = \max_{\mathcal{I}}(ALG(\mathcal{I})/OPT(\mathcal{I}))$ ($\alpha = \min_{\mathcal{I}}(ALG(\mathcal{I})/OPT(\mathcal{I}))$). For minimization (maximization) problems $\alpha \geq 1$ ($\alpha \leq 1$). The other commonly used convention is to define $\alpha = \min_{\mathcal{I}}(OPT(\mathcal{I})/ALG(\mathcal{I}))$ for a minimization problem (in which case, $\alpha \leq 1$).

7.2 Nondeterministic Rounding

7.2.1 Independent Rounding

In this section we illustrate the technique due to Raghavan and Thompson [3]. They developed the first constant-factor approximation algorithm for the minimum-width routing problem in two dimensions. Here we illustrate their technique on the Set Cover problem, basing our presentation on Vazirani [13]. Given a collection $S = \{S_1, S_2, \ldots, S_m\}$ of subsets of some universe $U = \{1, 2, \ldots, n\}$, the problem is to determine the minimum number of sets from S that cover all the elements of U. Let x_j be the variable associated with set S_j. Given below is the integer program IP and the corresponding linear programming relaxation LP for the set cover problem. The first constraint in the IP ensures that each element $i \in U$ is covered by some set in S, and the second constraint stipulates that the sets are picked integrally.

$$\begin{array}{lll} \text{IP:} & \text{minimize} & \sum_{j\in[1,m]} x_j \\ & \text{subject to:} & \sum_{j:i\in S_j} x_j \geq 1 \;\; \forall i \in U \\ & & x_j \in \{0, 1\} \;\; \forall i \in U \end{array} \qquad \begin{array}{lll} \text{LP:} & \text{minimize} & \sum_{j\in[1,m]} x_j \\ & \text{subject to:} & \sum_{j:i\in S_j} x_j \geq 1 \;\; \forall i \in U \\ & & 0 \leq x_j \leq 1 \;\; \forall i \in U \end{array}$$

Let x^* be the optimal solution to the linear programming relaxation above. In each iteration, we round each variable x_j to 1 with probability x_j^* and to 0 with probability $1 - x_j^*$. Each set S_j for which $x_j = 1$ is picked in the solution. The probability that element $i \in U$ is not covered in an iteration is $\prod_{j:i\in S_j}(1-x_j^*)$. If the element $i \in U$ occurs in the k sets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$, the values $x_{i_1}^*, x_{i_2}^*, \ldots, x_{i_k}^*$ are constrained by the inequality $\sum_{j=1}^{k} x_{i_j}^* \geq 1$, since the element $i \in U$ is covered by the optimal LP solution. The probability $\prod_{j=1}^{k}(1 - x_{i_j}^*)$ is then minimized when each value $x_{i_j}^*$ takes the value $1/k$. Thus, the probability that element $i \in U$ is not covered in an iteration is at least $(1-1/k)^k \geq 1/e$. Thus the probability that $i \in U$ is not covered after $c \log n$ iterations is $(1/e)^{c \log n} \leq 1/(4n)$ for some constant c. Equivalently, the probability that the solution computed after $c \log n$ iterations is not a valid cover is at most $\sum_{i=1}^{n} 1/(4n) = 1/4$. Furthermore, the expected number of sets in the solution computed is $(\sum_{j\in[1,m]} x_j^*)c \log n$. The probability that the number of sets is more than four times this expected value is at most 1/4 (follows from the Markov inequality). Therefore, with probability at least 1/2, the algorithm returns a cover with cost at most $(\sum_{j\in[1,m]} x_j^*)4c \log n$, implying that the performance ratio is $O(\log n)$. Srinivasan [14], observing that the constraints in the set cover problem are positively correlated, showed that the performance ratio of the randomized rounding algorithm is $\log(|U|/OPT) + O(\log\log(|U|/OPT)) + O(1)$.

Next, we consider an interesting idea due to Goemans and Williamson [4], in which two randomized rounding algorithms are run on each problem instance, and the better of the two is returned as the solution. This technique is used for the maximum satisfiability problem to obtain a 3/4-approximation algorithm, though each algorithm by itself does not provide a 3/4-approximation ratio. In the weighted version of the maximum satisfiability problem, we are given a Boolean formula in conjunctive normal form with weights on the clauses, and the goal is to determine an assignment of values (true/false) to the literals, such that the sum of weights of the clauses satisfied is maximized. The simpler rounding algorithm uses a purely randomized rounding, where each variable is set to true (false) with probability 1/2. If a clause j has k literals, then the probability that this clause is not satisfied is $1/2^k$ (corresponding to the situation when each of the k variables is set to 0). Thus, the probability that the clause is satisfied equals $1 - (1/2^k)$. To illustrate the second rounding algorithm (using linear programming) for this problem, we let C_j^+ denote the unnegated literals in the jth clause and C_j^- the negated literals in the jth clause in formula C. The integer program IP and the corresponding linear programming relaxation LP for the problem are given below.

$$\begin{array}{llll}
\text{IP:} & \text{maximize} \displaystyle\sum_{j \in C} w_j z_j & \qquad \text{LP:} & \text{maximize} \displaystyle\sum_{j \in C} w_j z_j \\
& \text{subject to:} & & \text{subject to:} \\
\displaystyle\sum_{i \in C_j^+} x_i + & \displaystyle\sum_{i \in C_j^-} (1 - x_i) \geq z_j \quad \forall j \in C & \displaystyle\sum_{i \in C_j^+} x_i + & \displaystyle\sum_{i \in C_j^-} (1 - x_i) \geq z_j \quad \forall j \in C \\
& z_j, x_i \in \{0, 1\} \quad \forall i, j & & 0 \leq z_j, x_i \leq 1 \quad \forall i, j
\end{array}$$

Let x^*, z^* be the optimal solution to the linear programming relaxation LP. The rounding sets literal i to true with probability x_i^* (without loss of generality we assume clause j contains only positive literals). The probability that clause j is satisfied after this rounding is $1 - \prod_{i \in C_j^+}(1 - x_i^*)$. If clause j contains k literals, $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$, the values $x_{i_1}^*, x_{i_2}^*, \ldots, x_{i_k}^*$ are constrained by the inequality $\sum_{j=1}^{k} x_{i_j}^* \geq z_j^*$, a constraint in the LP formulation. The probability $\prod_{j=1}^{k}(1 - x_{i_j}^*)$ is then maximized when each value $x_{i_j}^*$ takes the value z_j^*/k. Thus, the probability that clause j is not satisfied after the rounding is at most $(1 - z_j^*/k)^k$. Thus the probability that clause j is satisfied after the rounding is at least $1 - (1 - z_j^*/k)^k$. Observing that $1 - (1 - z_j^*/k)^k \geq z_j^*(1 - (1 - 1/k)^k)$ for $0 \leq z_j^* \leq 1$ (due to concavity), the probability that clause j is satisfied is at least $z_j^*(1 - (1 - 1/k)^k)$.

The bound of 3/4 follows from the fact that for each clause j, $\max\{(1 - 1/2^k), z_j^*(1 - (1 - 1/k)^k)\} \geq \max\{z_j^*(1 - 1/2^k), z_j^*(1 - (1 - 1/k)^k)\} \geq \frac{z_j^*}{2}\{(1 - 1/2^k) + (1 - (1 - 1/k)^k)\} \geq 3/4 z_j^*$ for every positive integer k.

7.2.2 Dependent Rounding

7.2.2.1 Simultaneous Rounding

The idea of simultaneously rounding a set of variables was used by Bertsimas et al. [5] to establish the integrality of several well-known polytopes. In particular, they established the integrality of the polytopes associated with the minimum $s - t$ cut, p-median on a cycle, uncapacitated lot sizing, and boolean optimization. Using this technique, Bertsimas et al. [5] established a bound of $2(1-1/2^k)$ for the minimum k-sat problem. A bound of 2 is established for the feasible cut problem, by showing it is equivalent to vertex cover, which is approximable within a factor of 2 [15]. Here we illustrate the technique due to Bertsimas et al. [5] on the feasible cut problem. This technique is particularly interesting as the analysis of the performance ratio is considerably simplified.

Given a graph $G = (V, E)$ with weights on the edges, M a set of pairs of nodes in G, and a source vertex s. The problem is to determine a cut of minimum weight with the additional constraints that s belongs to the cut, but for any pair $(i, j) \in M$, both i and j are not in the cut. The integer program IP for the feasible cut problem and the corresponding linear programming relaxation LP are given below.

$$\begin{array}{lll@{\qquad}lll}
\text{IP:} & \text{minimize} & \sum_{(i,j) \in E} c_{ij} x_{ij} & \text{LP:} & \text{minimize} & \sum_{(i,j) \in E} c_{ij} x_{ij} \\
& \text{subject to:} & x_{ij} \geq y_i - y_j \quad \forall (i, j) \in E & & \text{subject to:} & x_{ij} \geq y_i - y_j \quad \forall (i, j) \in E \\
& & x_{ij} \geq y_j - y_i \quad \forall (i, j) \in E & & & x_{ij} \geq y_j - y_i \quad \forall (i, j) \in E \\
& & y_i + y_j \leq 1 \quad \forall (i, j) \in M & & & y_i + y_j \leq 1 \quad \forall (i, j) \in M \\
& & y_s = 1 & & & y_s = 1 \\
& & x_{ij}, y_j \in \{0, 1\} \quad \forall i, j & & & 0 \leq x_{ij}, y_j \leq 1 \quad \forall i, j
\end{array}$$

In this technique the variables are rounded simultaneously with respect to a random variable. Let U be a random value in $[1/2, 1]$ generated uniformly. Given an optimal solution (x^*, y^*) to the linear program LP, construct the cut as follows: if $y_i^* < U$ then $y_i = 0$, and if $y_i^* > U$ then $y_i = 1$. The rounding operation gives a feasible cut, since for each $(i, j) \in M$ at most one of y_i^*, y_j^* is greater than 1/2. Let Z_{IP}

be the value of the optimal solution to IP, Z_{LP} the value of the optimal solution to LP, and $E(Z_R)$ the expected value of the solution obtained after rounding.

Theorem 7.1

Minimum feasible cut can be approximated within a factor of 2.

Proof

Clearly, $Z_{IP} \leq E(Z_R)$. We show that $E(Z_R) \leq 2Z_{LP}$. If $E(x_{ij}) \leq 2x^*_{ij}$ for all i, j, then by linearity of expectation the result holds. Without loss of generality assume that $y^*_i \leq y^*_j$. If $y^*_j \leq 1/2$ then $E(x_{ij}) = 0$. If $y^*_i \leq 1/2 \leq y^*_j$, then $E(x_{ij}) = P(U \in [1/2, y^*_j]) = 2(y^*_j - 1/2) \leq 2(y^*_j - y^*_i)$. If $y^*_i \geq 1/2$, then $E(x_{ij}) = 2(y^*_j - y^*_i)$. This implies that $E(x_{ij}) \leq 2(y^*_j - y^*_i) \leq 2x^*_{ij}$. □

7.2.2.2 Rounding against a Hyperplane

The first substantial improvement for the Max-Cut problem was made by Goemans and Williamson [6], who presented a 0.87856 factor approximation based on semidefinite programming. The above bound is also applicable to the Max 2-Sat problem. They also gave a 0.7584 factor approximation algorithm for the Max Sat problem. Here we outline their technique for the Max-Cut problem. Given a graph $G = (V, E)$ with weights on the edges, the objective is to partition the vertices of G such that the sum of weights of the cut edges is maximized. The problem is formulated first as a quadratic (nonlinear) program, and a relaxation of the quadratic program is defined in which each variable corresponds to a vector. An optimal solution to this relaxed nonlinear program is then computed. Given a random hyperplane, the vertices are partitioned into two sets, corresponding to points above and below the hyperplane. This partition has the desired bound. For details of the proof and the algorithm for computing the optimal solution to the relaxed program VP, we refer the reader to Vazirani [13] and Chapter 8 on Semidefinite Programming by Ye, So, and Zhang. Next, we describe their formulations and the randomization procedure.

$$\begin{array}{lll} \text{QP:} & \text{maximize} & 1/2\sum_{(i,j)\in E} w_{ij}(1-y_iy_j) \\ & \text{subject to:} & y_i^2 = 1 \quad \forall i \in V \\ & & y_i \in \mathcal{Z} \quad \forall i \in V \end{array} \qquad \begin{array}{lll} \text{VP:} & \text{maximize} & 1/2\sum_{(i,j)\in E} w_{ij}(1-v_iv_j) \\ & \text{subject to:} & v_i \cdot v_i = 1 \quad \forall i \in V \\ & & v_i \in \mathcal{R}^n \quad \forall i \in V \end{array}$$

Let r be a uniformly distributed vector in unit sphere S_{n-1}, then $S = \{i : v_i \cdot r \geq 0\}$ and $V \setminus S$ are the two sets defining the partition.

7.2.2.3 Extensions

Next we outline some extensions of the basic rounding technique. In all these techniques, the variables are rounded randomly in a somewhat dependent fashion. First we consider the assignment problem in the presence of covering constraints. Given a complete bipartite graph $G = (A \cup B, E)$, with $|A| = |B|$, and weights on the edges. The objective is to find a matching of minimum weight that satisfies the covering constraints. The integer program IP and the linear programming relaxation LP for the assignment problem are given below.

$$\begin{array}{ll} \text{IP: minimize} & \sum_{(i,j)\in E} c_{ij}x_{ij} \\ \quad\text{subject to:} & \sum_{j\in B} x_{ij} = 1 \ \forall i \in A \\ & \sum_{i\in A} x_{ij} = 1 \ \forall j \in B \\ & \sum_{i\in A, j\in B} a^k_{ij}x_{ij} \geq b^k \ \forall k \in [1, K] \\ & x_{ij}, y_j \in \{0, 1\} \ \forall i \in A, \ j \in B \end{array} \qquad \begin{array}{ll} \text{LP: minimize} & \sum_{(i,j)\in E} c_{ij}x_{ij} \\ \quad\text{subject to:} & \sum_{j\in B} x_{ij} = 1 \ \forall i \in A \\ & \sum_{i\in A} x_{ij} = 1 \ \forall j \in B \\ & \sum_{i\in A, j\in B} a^k_{ij}x_{ij} \geq b^k \ \forall k \in [1, K] \\ & 0 \leq x_{ij}, y_j \leq 1 \ \forall i \in A, \ j \in B \end{array}$$

In the absence of the covering constraint ($\sum_{i\in A, j\in B} a^k_{ij}x_{ij} \geq b^k$), the polytope associated with the IP is integral. But in the presence of the covering constraints we can only guarantee a fractional optimal solution to the LP in polynomial time. One possibility is to obtain an integral solution by rounding [3] the optimal fractional solution. One major difficulty with independent rounding in the presence of equality constraints is that the probability that the constraint is satisfied could be as low as $1/e$ (consider the case when all the $x_{ij}s$ have the same value $1/|A|$). Therefore, the expected number of equality constraints satisfied in one rounding iteration is low. However, the covering constraints are satisfied almost approximately. Arora et al. [7] developed a randomized rounding technique that obtains an integral solution (from the fractional solution), which satisfies $(|A| - o(|A|)$ equality constraints and all the covering constraints almost approximately $(\sum_{i\in A,\, j\in B} a^k_{ij}x_{ij} \geq b^k - O(\sqrt{|A|}\max\{a^k_{ij}\}))$. Next, we describe their rounding algorithm for the case when all the fractional values are constants. For the rounding in the general case, and for the proofs we refer the reader to the original paper. Let x^* be the optimal fractional solution. The algorithm first constructs a multigraph from the bipartite graph as follows: for each edge in G, toss a biased coin (with probability of head x^*_{ij}) $\Theta(\log^3(n))$ times. If heads show up a times then the multigraph has a copies of edge (i, j). The multigraph is a union of paths and cycles of length $O(\sqrt{n})$ (if not then we have to delete $O(\sqrt{n})$ edges). Now these paths and cycles are further divided into $\Theta(\sqrt{n})$ groups of size $O(\sqrt{n})$ each. Within each group, either all the edges of A are picked or all the edges of B are picked, and the decision is equally likely. Using a generalization of this technique, Arora et al. [7] were able to demonstrate polynomial-time approximation schemes for dense instances of minimum linear arrangement problem, minimum cut linear arrangement problem, maximum acyclic subgraph problem, and the betweenness problem.

Next we briefly mention some other techniques. Srinivasan [16] developed a rounding technique based on distributions on level sets, and established better approximation ratios for low-congestion multipath routing problem, and the maximum coverage version of set cover problem. Gandhi et al. [17] developed a new rounding scheme based on the pipage rounding method of Ageev and Sviridenko [10] (see Section 7.3.4), and the level set-based method of Srinivasan [16] to obtain better approximation algorithms for the throughput maximization problem in broadcast scheduling, the delay minimization problem in broadcast scheduling, and the capacitated vertex cover problem. Another dependent rounding technique has been developed by Doerr [18], with applications to digital halftoning. Doerr [19] developed another dependent randomized rounding technique that respects cardinality constraints.

7.3 Deterministic Rounding

7.3.1 Scaling

Scaling is an important technique that has been applied to covering problems such as Vertex Cover to obtain a simple 2-factor approximation. Our presentation is based on Hochbaum [12] (Chapter 3). Given that it is still not known whether vertex cover admits an approximation ratio strictly better (by a constant) than 2, scaling seems to be a powerful technique. Given a graph $G = (V, E)$ with weights on the vertices. The objective is to determine a minimum-weight set $S \subset V$, such that every edge has at least one endpoint in S. Given below is the integer program IP and the corresponding linear programming relaxation LP.

$$\begin{array}{llll} \text{IP:} & \text{minimize} & \sum_{i\in V} w_i x_i & \\ & \text{subject to:} & x_i + x_j \geq 1 & \forall (i, j) \in E \\ & & x_i \in \{0, 1\} & \forall i \in V \end{array} \qquad \begin{array}{llll} \text{LP:} & \text{minimize} & \sum_{i\in V} w_i x_i & \\ & \text{subject to:} & x_i + x_j \geq 1 & \forall (i, j) \in E \\ & & 0 \leq x_i \leq 1 & \forall i \in V \end{array}$$

Let x^* be the optimal solution to the linear program LP. Let S be the set of vertices j such that $x^*_j \geq 1/2$. S is a cover because for each edge (i, j) either x_i or x_j is $\geq 1/2$, and the weight of S is at most $2\sum_{i\in V} w_i x^*_i$. Interestingly, the algorithm by Gonzalez [20] is the only factor 2 approximation algorithm for vertex cover, whose proof does not rely on the theory of linear programming.

7.3.2 Filter and Round

Sahni and Gonzalez [21] showed that for certain problems including the p-median problem, the tree pruning problem, and the generalized assignment problem, finding an α-approximate solution is NP-hard. In light of the previous result, the next best thing is to find an α-approximate solution with the minimum number of constraint violations. Lin and Vitter [8] gave such approximation algorithms for the problems mentioned above. For the generalized assignment problem, we refer the reader to Chapter 48 by Yagiura and Ibaraki. Here we will illustrate their technique on the p-median problem. Our presentation is based on Lin and Vitter [8]. Given a complete graph G on n vertices with weights on the edges and an integer p, the problem is to determine p vertices (medians) so that the sum of the distances from each vertex to its closest median is minimized. The integer program IP and the corresponding linear programming relaxation LP are given below.

$$\begin{array}{lll}
\text{IP:} & \text{minimize} & \sum_{i,j\in V} c_{ij}x_{ij} \\
& \text{subject to:} & \sum_{j\in V} x_{ij} = 1 \quad \forall i \in V \\
& & x_{ij} \le y_j \quad \forall i, j \in V \\
& & \sum_{j\in V} y_j = p \\
& & x_{ij}, y_j \in \{0, 1\} \quad \forall i, j
\end{array}
\qquad
\begin{array}{lll}
\text{LP:} & \text{minimize} & \sum_{i,j\in V} c_{ij}x_{ij} \\
& \text{subject to:} & \sum_{j\in V} x_{ij} = 1 \quad \forall i \in V \\
& & x_{ij} \le y_j \quad \forall i, j \in V \\
& & \sum_{j\in V} y_j = p \\
& & 0 \le x_{ij}, y_j \le 1 \quad \forall i, j
\end{array}$$

Given an optimal solution x^*, y^* to the LP, we obtain an integer program FP (called a filtered program) by setting some variables in x to 0. The FP has the property that any integral feasible solution is at most $(1+\alpha)$ times the value of the optimal solution to LP. First, a fractional feasible solution to FP is constructed from x^*, y^*. A feasible integral solution to FP is then obtained using either randomized rounding or some greedy rounding. Here we illustrate a deterministic (greedy) rounding method. We assume certain lemmas to illustrate the technique. For the proof of the lemmas, we refer the reader to the original paper by Lin and Vitter [8].

Lemma 7.1

Given y, the optimal values for x can be computed for the linear programming problem LP.

Given an optimal solution x^*, y^* to the LP, for a vertex $i \in V$, let V_i be the set of vertices j such that $c_{ij} \le (1+\alpha)\sum_{j\in V} c_{ij}x^*_{ij}$. The FP and the reduced filtered program (RFP) necessary to compute the solution to FP by Lemma 7.1 are

$$\begin{array}{lll}
\text{FP:} & \text{minimize} & L \\
& \text{subject to:} & \sum_{j\in V_i} x_{ij} = 1 \quad \forall i \in V \\
& & x_{ij} \le Ly_j \quad \forall i, j \in V \\
& & \sum_{j\in V} y_j = p \\
& & x_{ij} = 0 \quad \forall i \in V,\ j \in V\backslash V_i \\
& & x_{ij}, y_j \in \{0, 1\} \quad \forall i, j
\end{array}
\qquad
\begin{array}{lll}
\text{RFP:} & \text{minimize} & \sum_{j\in V} y_j \\
& \text{subject to:} & \sum_{j\in V_i} y_j \ge 1 \quad \forall i \in V \\
& & y_j \in \{0, 1\} \quad \forall i, j
\end{array}$$

L corresponds to the factor by which the covering constraints are violated. The following lemma holds by construction.

Lemma 7.2

Any feasible (integral) solution to FP has value at most $(1+\alpha)$ times the value of the optimal solution to the linear programming relaxation LP.

It is the case that $\sum_{j\in V_i} y^*_j \ge \alpha/(1+\alpha)$. Therefore, a feasible fractional solution to RFP with value $(1+1/\alpha)p$ can be constructed (by assigning $y_j = y^*_j(1+\alpha)/\alpha$). RFP is nothing but set cover, and a $\log n$ approximate integral solution can be constructed using the greedy heuristic of Chvátal [22]. Therefore, by Lemma 7.2 we have a $(1+\alpha)p\log n$ approximate constraint violations with value at most $(1+\alpha)$ times the value of the optimal solution to the integer program.

7.3.3 Iterated Rounding

The technique of iterated rounding was introduced by Jain [9], who gave a 2-factor approximation algorithm for the generalized Steiner network problem. Consider the problem of finding a minimum-cost edge-induced subgraph of a graph that contains a prespecified number of edges from each cut. Formally, $G = (V, E)$ is a graph with weights on the edges. Also given is a function $f : 2^V \rightarrow \mathcal{Z}$. The problem is to determine a minimum-weight set of edges such that for every R subset of V, the number of edges is $\delta(R) \geq f(R)$, where $\delta(R)$ are the edges in the cut defined by the vertices in R. Given below are the integer program IP and the corresponding linear programming relaxation LP.

$$\begin{array}{lll} \text{IP:} & \text{minimize} & \sum_{e\in E} w_e x_e \\ & \text{subject to:} & \sum_{e\in\delta(R)} \geq f(R) \quad \forall S \subseteq V \\ & & x_e \in \{0, 1\} \quad \forall i \in E \end{array} \qquad \begin{array}{lll} \text{LP:} & \text{minimize} & \sum_{e\in E} w_e x_e \\ & \text{subject to:} & \sum_{e\in\delta(R)} \geq f(R) \quad \forall S \subseteq V \\ & & 0 \leq x_e \leq 1 \quad \forall i \in E \end{array}$$

Note that both the programs above contain exponentially many constraints. Jain [9] gives a separation oracle for the linear programming relaxation. Using this separation oracle, an optimal solution can be computed in polynomial time [23]. Furthermore, Jain establishes the following:

Theorem 7.2

Any basic feasible solution to the linear programming relaxation has at least one variable with value $\geq 1/2$.

Based on the previous theorem, one can construct a solution as follows: find an optimal solution (basic) to the LP, include all the edges with a value $\geq 1/2$ in the solution, then recursively solve the subproblem obtained by deleting the edges included in the solution.

7.3.4 Pipage Rounding

Pipage rounding was developed by Ageev and Sviridenko [10], who applied it to the maximum coverage problem, hypergraph maximum k-cut with given sizes of parts, and scheduling on unrelated parallel machines. They showed that the maximum coverage problem can be approximated within $1 - (1 - 1/k)^k$ where k is the maximum size of any subset, thereby improving the previous bound of $1 - 1/e$ due to Cornuejols et al. [24]. For the hypergraph max k-cut they obtained a bound of $1 - (1 - 1/r)^r - 1/r^r$, where r is the cardinality of the smallest edge in the hypergraph. For the scheduling problem on unrelated machines, they considered an additional constraint on the number of jobs that a given machine can process and obtained the bound of 3/2. A similar bound was also established by Skutella [25] in the absence of cardinality constraints. For the case of two machines, the current best bound is 1.2752 due to Skutella [25], obtained by rounding the semidefinite programming relaxation using the dependent rounding technique of Goemans and Williamson [6]. Ageev et al. [26] obtained a 1/2-approximation algorithm for the max-dicut problem with given sizes of parts by a refined application of the pipage rounding. Recently, Galluccio and Nobili [27] have improved the approximation ratio from 3/4 to $1 - 1/2q$ for the maximum coverage problem when all the sets are of size 2, where every clique in a clique cover of the input graph has size at least q. Note that $q \geq 2$. This problem is also known as the maximum vertex cover problem. Pipage rounding is especially suited to problems involving assignment and cardinality constraints.

Our description of the pipage rounding is based on Ageev and Sviridenko [10]. The idea is to deterministically round a fractional solution to an integral solution, while ensuring that the objective function value does not decrease in the rounding process. If the starting fractional solution was at least c times the optimal fractional solution, then the pipage rounding will guarantee a c-approximation algorithm. The rounding process converts a fractional solution into another fractional solution with less number of nonintegral components. The "δ-convexity" of the objective functions guarantees that the objective function value does not decrease in the rounding process.

Let $G = (V, E)$ be a bipartite graph with capacities c_v on the vertices. Let $f(X)$ be a polynomially computable function defined on the values $X = \{x_e : e \in E\}$ assigned to the edges of G. Consider the following integer program IP whose solution is an assignment of 0, 1 to the edges that maximizes $f(X)$

subject to the capacity constraints, and its linear programming relaxation LP:

$$\begin{array}{llll} \text{IP:} & \text{maximize} & f(X) & \\ & \text{subject to:} & \sum_{e\in N(v)} x_e \le c_v & \forall v \in V \\ & & x_e \in \{0,1\} & \forall e \in E \end{array} \qquad \begin{array}{llll} \text{LP:} & \text{maximize} & f(X) & \\ & \text{subject to:} & \sum_{e\in N(v)} x_e \le c_v & \forall v \in V \\ & & 0 \le x_e \le 1 & \forall e \in E \end{array}$$

We do not assume that the optimal solution to the LP is computable in polynomial time. Given a fractional solution X, let $G(X)$ be the subgraph induced by the edges that are assigned a nonintegral value in X. $G(X)$ either contains a path P or a cycle C. Let $P_o(C_o)$ be the odd-indexed edges in the $P(C)$. Similarly, $P_e(C_e)$ the set of even-indexed edges in $P(C)$. Given $P(C)$, let $lb = \min\{\min\{x_e : e \in P_o(C_o)\}, \min\{1 - x_e : e \in P_e(C_e)\}\}$. Similarly, define $ub = \min\{\min\{1 - x_e : e \in P_o(C_o)\}, \min\{x_e : e \in P_e(C_e)\}\}$. f is said to be δ-convex with respect to $\delta \in [lb, ub]$ if for each fractional solution and all paths and cycles it is convex in δ. Given δ-convexity, the maximum of f in $[lb, ub]$ is attained at one of the endpoints. Pipage rounding amounts to either successively adding and deleting ub, or successively deleting and adding lb, from the values assigned to the edges in $P(C)$. This process yields a solution with a reduced number of nonintegral components. Let us examine the case when all the capacities are 1, and f computes the sum of the values assigned to the edges. In this case, the solution to the IP corresponds to a maximum matching, and the solution to the linear program corresponds to the maximum fractional matching. The pipage rounding (as can be readily verified) in this case converts the fractional matching into an integral matching of same or larger size. To compute an α-approximation it remains to find a function g that approximates f within α such that maximum of g can be computed in polynomial time, subject to the constraints in the LP.

We illustrate the application of pipage rounding to the maximum coverage problem, where we are given a collection S of weighted subsets of ground set I, and an integer k. The goal is to determine $X \subseteq I$ of cardinality k such that the sum of the weights of the sets in S that intersect with X is maximized. Associated with each element $i \in I$ is a variable x_i, and associated with each element S_j of S is a variable z_j. Given below is an integer program for the maximum coverage problem.

$$\begin{array}{lll} \text{IP:} & \text{maximize} & \sum_{j=1}^{m} w_j z_j \\ & \text{subject to:} & \sum_{i\in S_j} x_i \ge z_j, \quad \forall S_j \in S \\ & & \sum_{i=1}^{n} x_i = k \\ & & x_i \in \{0,1\} \quad \forall i \in I \end{array}$$

The objective function in IP above can be replaced with $f = \sum_{j=1}^{m} w_j(1 - \prod_{i\in S_j}(1 - x_i))$ as it has the same value over all integral vectors x. Replace f by $g = \sum_{j=1}^{m} w_j \min\{1, \sum_{i\in S_j} x_i\}$. It can be shown that f and g are δ-convex and g approximates f within a factor of $1-(1-1/k)^k$, where k is the cardinality of the largest element of S. Furthermore, the fractional optimal solution to g, subject to the constraints in IP can be computed in polynomial time.

7.3.5 Decompose and Round

We next describe a deterministic technique due to Gaur et al. [11]. This technique is applicable to geometric covering problems, and can be thought of as an extension of the scaling technique. We consider covering problems of the form min cx, subject to $Ax \ge b$, $x \in \{0,1\}^n$, where A is an $m \times n$ matrix with 0, 1 entries, and c, x are vectors of dimension n and b is a vector with m entries. The geometry of the problem under consideration imposes a structure on A, and this helps us in the application of the scaling technique. We begin with a few definitions. Let $C = \{1, \ldots, n\}$ be the set of indices of the columns in A and $R = \{1, \ldots, m\}$ the set of indices of the rows in A. Denote by $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ a partition of R, and by $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ a partition of the columns of A. $A(R_i, C_j)$ is the matrix obtained from A by removing the columns in $C \setminus C_j$, and the rows in $R \setminus R_i$. A matrix A is *totally unimodular* if the determinant of every square submatrix of A is ±1. We say A is *partially unimodular* with respect to $\mathcal{C}$ and $\mathcal{R}$ if for all $C_i \in \mathcal{C}$, $R_j \in \mathcal{R}$, $A(R_j, C_i)$ is totally unimodular. For a partially unimodular matrix A,

$|\mathcal{C}| = |\mathcal{R}|$ is also known as the *partial width* of A. It is well known that if M is block structured and if all the blocks are totally unimodular then M is totally unimodular. This fact, with a suitable reordering of the rows and columns, implies the following:

Lemma 7.3

Let A_D be the matrix whose ith diagonal block corresponds to $A(R_i, C_i)$ (all other entries are 0) then A_D is totally unimodular, if A is partially unimodular with respect to $\mathcal{R}$, $\mathcal{C}$.

We next describe the rectangle stabbing problem, and show that its coefficient matrix is partially unimodular and has partial width 2. A $\log n$ factor approximation for the rectangle stabbing problem is due to Hassin and Megiddo [28]. Given a set of axis-aligned rectangles (in 2D), the problem is to determine the minimum number of axis-parallel lines that are needed to stab all the rectangles. Let H be the set of horizontal lines going through the horizontal edges of the rectangles, V the set of vertical lines going through the vertical edges of the rectangles, and R the set of all rectangles. Let $H_r(V_r)$ be the set of lines from $H(V)$ that intersect rectangle $r \in R$. Given below is the integer program IP and the corresponding linear programming relaxation LP.

$$\begin{array}{lll} \text{IP: minimize} & \sum_{i\in H} h_i + \sum_{j\in V} v_j \\ \text{subject to:} & \sum_{i:i\in H_r} h_i + \sum_{j:j\in V_r} v_j \geq 1 \;\; \forall r\in R \\ & h_i, v_j \in \{0, 1\} \;\; \forall i \in H,\ j \in V \end{array}$$

$$\begin{array}{lll} \text{LP: minimize} & \sum_{i\in H} h_i + \sum_{j\in V} v_j \\ \text{subject to:} & \sum_{i:i\in H_r} h_i + \sum_{j:j\in V_r} v_j \geq 1 \, \forall r\in R \\ & 0 \leq h_i, v_j \leq 1 \;\; \forall i \in H,\ j \in V \end{array}$$

Let A be the coefficient matrix corresponding to the programs above.

Lemma 7.4

A is partially unimodular with respect to $\mathcal{C} = \{H, V\}$ and $\mathcal{R} = \{R_h, R_v\}$ as computed below.

Given an optimal solution h^*, v^* to the linear programming relaxation LP, we construct a partition $\mathcal{R} = \{R_h, R_v = R \backslash R_h\}$ of the rectangles of R as follows: R_h is the set of all the rectangles r such that $\sum_{i:i\in H_r} h_i \geq 1/2$. Let A_D be the block diagonal matrix whose blocks are $A(R_h, H)$ and $A(R_v, V)$. $A(R_h, H)$ and $A(R_v, V)$ are totally unimodular as the columns can be reordered so that each row has the consecutive ones property. By Lemma 7.3, A_D is totally unimodular. Consider the program min cx subject to $A_D x \geq 1$, $x \in \{0, 1\}^n$. Conforti et al. [29] showed that the polytope associated with A_D is integral, hence the optimal integral solution has the same value as the optimal fractional solution. Note that $(2h^*, 2v^*)$ is feasible in the previous problem. Therefore, the performance ratio is 2 as $ALG \leq (2h^*, 2v^*)$ and $OPT \geq (h^*, v^*)$. Furthermore, the addition of capacity constraints on H and V does not affect the performance ratio. These results can be generalized for arbitrary weights on the lines and requirements on the rectangles in d dimensions. For recent results on the rectangle stabbing problem with soft capacities see Even et al. [30]. The case when rectangles have zero height has been studied extensively, see Chapter 37 by Kovaleva and Spieksma.

A brief comment about the technique is in order. Every matrix A is partially unimodular with respect to the following partitions: $C_1, C_2, \ldots, C_n$, where C_i is the ith column in A. Let x^* be the optimal solution to the LP. Consider the following partition of rows: rectangle r belongs to set R_i in the partition if $x_i^* A[r, i] = \max_{j\in[1,\ldots,n]}\{x_j^* A[r, j]\}$. A_D can now be constructed from the blocks $A(R_i, C_i)$. Once again, by Lemma 7.3 A_D is totally unimodular as each $A(R_i, C_i)$ is a column vector with all ones (the determinant for every square submatrix is 1) and totally unimodular. Let τ be the maximum number of nonzero entries in a row of A. The performance ratio using the algorithm and the argument above is $1/\tau$. This is similar to the bound obtained for the set cover problem using the scaling technique. In this sense, our approach can be viewed as a generalization of the scaling technique.

The arguments outlined in the preceding paragraphs lead to the following theorem.

Theorem 7.3

Given a covering problem of the form min cx, subject to $Ax \geq b$, $x \in \{0, 1\}^n$, if A is partially unimodular and has partial width $1/\alpha$, there exists an approximation algorithm with performance ratio α.

In light of the preceding theorem it is natural to study algorithms (exact and approximation) for determining the minimum cardinality partitions with respect to which A is partially unimodular. We are not aware of any existing results and pose the determination of minimum partial width as an interesting open problem, with application to the theory of approximation algorithms.

Next, we consider an application of the rectangle stabbing problem to a load balancing problem that arises in the context of scheduling on multiprocessor systems. In the rectilinear partitioning problem, the input is a matrix of integers, and the problem is to partition the matrix using h horizontal lines and v vertical lines, such that the load inside each rectangle (formed by two consecutive horizontal and vertical lines) is minimized, where the load of a rectangle is defined to be the sum of entries in the rectangle. Given an instance of the rectilinear partitioning problem we construct an instance of the rectangle stabbing problem as follows: let L be the minimum load (we can obtain this by using binary search), all the submatrices with load in excess of L correspond to rectangles in the rectangle stabbing problem. Note that if all the rectangles are stabbed then the load is at most L. As we only have a 2-factor approximation algorithm for the rectangle stabbing problem, the number of lines returned can be twice the number of lines stipulated. Therefore, a solution to the rectilinear partitioning problem is obtained by removing every second line (horizontal as well as vertical). In the process of removing the alternate lines, a new rectangle is formed whose load is at most $4L$. Therefore, the performance ratio is 4.

7.4 Discussion

Numerous techniques have been developed over the last two decades to convert an optimal fractional solution (to the linear programming relaxation of an integer program) to an approximate integral solution. These techniques can be divided into two broad categories: those that use randomized strategies and ones that use deterministic strategies. Most of the randomized strategies can be made deterministic (at the expense of increased running time) using the method of conditional expectation. The applicability of the strategies is most evident in the context of packing and covering types of problems. Some success has been obtained in the application of these techniques in the presence of cardinality constraints.

References

[1] Erdős, P. and Selfridge, J. L., On a combinatorial game, *J. Comb. Theory Ser. A*, 14, 298, 1973.

[2] Alon, N. and Spencer, J. H., *The Probabilistic Method*, Wiley-Interscience, New York, 2000.

[3] Raghavan, P. and Thompson, C. D., Randomized rounding: a technique for provably good algorithms and algorithmic proofs, *Combinatorica*, 7(4), 365, 1987.

[4] Goemans, M. X. and Williamson, D. P., New $\frac{3}{4}$-approximation algorithms for the maximum satisfiability problem, *SIAM J. Disc. Math.*, 7(4), 656, 1994.

[5] Bertsimas, D., Teo, C., and Vohra, R., On dependent randomized rounding algorithms, *Oper. Res. Lett.*, 24(3), 105, 1999.

[6] Goemans, M. X. and Williamson, D. P., Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *JACM*, 42(6), 1115, 1995.

[7] Arora, S., Frieze, A., and Kaplan, H., A new rounding procedure for the assignment problem with applications to dense graph arrangement problems, *Math. Prog. Ser. A*, 92(1), 1, 2002.

[8] Lin, J. H. and Vitter, J. S., Approximation algorithms for geometric median problems, *Inf. Proc. Lett.*, 44(5), 245, 1992.

[9] Jain, K., A factor 2 approximation algorithm for the generalized Steiner network problem, *Combinatorica*, 21(1), 39, 2001.

[10] Ageev, A. A. and Sviridenko, M. I., Pipage rounding: a new method of constructing algorithms with proven performance guarantee, *J. Comb. Optim.*, 8(3), 307, 2004.

[11] Gaur, D. R., Ibaraki, T., and Krishnamurti, R., Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem, *J. Algorithms*, 43(1), 138, 2002.

[12] Hochbaum, D. S., Ed., *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Co., Boston, MA, 1997.

[13] Vazirani, V. V., *Approximation Algorithms*, Springer, Berlin, 2001.

[14] Srinivasan, A., Improved approximation guarantees for packing and covering integer programs, *SIAM J. Comput.*, 29(2), 648, 1999.

[15] Yu, B. and Cheriyan, J., Approximation algorithms for feasible cut and multicut problems, *Proc. of ESA*, LNCS, 979, 1995.

[16] Srinivasan, A., Distributions on level-sets with applications to approximation algorithms, *Proc. of FOCS,* 2001, 588.

[17] Gandhi, R., Khuller, S., Parthasarathy, S., and Srinivasan, A., Dependent rounding in bipartite graphs, *Proc. of FOCS*, 2002, 323.

[18] Doerr, B., Nonindependent randomized rounding and an application to digital halftoning, *SIAM J. Comput.*, 34(2), 299, 2005.

[19] Doerr, B., Roundings respecting hard constraints, *Proc. of STACS*, 2005, 617.

[20] Gonzalez, T. F., A simple LP-free approximation algorithm for the minimum weight vertex cover problem, *Inform. Proc. Lett.*, 54(3), 129, 1995.

[21] Sahni, S. and Gonzalez, T. F., P-complete approximation problems, *JACM*, 23(3), 555, 1976.

[22] Chvátal, V., A greedy heuristic for the set-covering problem, *Math. Oper. Res.*, 4(3), 233, 1979.

[23] Grötschel, M., Lovász, L., and Schrijver, A., Geometric algorithms and combinatorial optimization, in *Algorithms and Combinatorics*, 2nd ed., Springer-Verlag, Berlin, 1993.

[24] Cornuejols, G., Fisher, M. L., and Nemhauser, G. L., Location of bank accounts to optimize float: an analytic study exact and approximate algorithms, *Manage. Sci.*, 23, 789, 1977.

[25] Skutella, M., Convex quadratic and semidefinite programming relaxations in scheduling, *JACM*, 48(2), 206, 2001.

[26] Ageev, A., Hassin, R., and Sviridenko, M., An 0.5-approximation algorithm for MAX DICUT with given sizes of parts, *SIAM J. Disc. Math.*, 14(2), 246, 2001.

[27] Galluccio, A. and Nobili, P. Improved approximation of maximum vertex cover, *Oper. Res. Lett.* 34(1), 72–84, 2006.

[28] Hassin, R. and Megiddo, N., Approximation algorithms for hitting objects with straight lines, *Disc. Appl. Math.*, 30(1), 29, 1991.

[29] Conforti, M., Cornuéjols, G., and Truemper, K., From totally unimodular to balanced 0, ±1 matrices: a family of integer polytopes, *Math. Oper. Res.*, 19(1), 21, 1994.

[30] Even, G., Rawitz, D., and Shahar, S., Approximation of rectangle stabbing with soft capacities, in *Workshop on Interdisciplinary Application of Graph Theory, Combinatorics, and Algorithms*, 2005.

8

On Analyzing Semidefinite Programming Relaxations of Complex Quadratic Optimization Problems

Anthony Man-Cho So
Stanford University

Yinyu Ye
Stanford University

Jiawei Zhang
New York University

8.1 Introduction

Following the seminal work of Goemans and Williamson [1], there has been an outgrowth in the use of semidefinite programming (SDP) for designing approximation algorithms. Recall that an α-approximation algorithm for a problem $\mathscr{P}$ is a polynomial-time algorithm such that for every instance I of $\mathscr{P}$, it delivers a solution that is within a factor of α of the optimum value [2]. It is well known that SDPs can be solved in polynomial time (up to any prescribed accuracy) via interior-point algorithms (see, e.g., Refs. [3,4]), and they have been used very successfully in the design of approximation algorithms for a host of NP-hard problems, e.g., graph partitioning, graph coloring, and quadratic optimization [1–9], just to name a few.

Before we delve into the main topics of this chapter, let us first review Goemans and Williamson's technique of analyzing SDP relaxations and point out its limitations. Consider the following (real) discrete quadratic programming (QP) problem:

$$\begin{aligned} &\text{maximize} \quad \sum_{i,j} Q_{ij}(1 - x_i x_j) \\ &\text{subject to} \quad x_k \in \{-1, 1\}, \quad k = 1, 2, \ldots, n \end{aligned} \tag{8.1}$$

where Q is an $n \times n$ symmetric, positive-semidefinite matrix. Problem (8.1) captures a wide variety of combinatorial optimization problems (e.g., MAX CUT), and is known to be NP-hard. It is thus natural to search for a relaxation of problem (8.1) that is polynomial-time solvable and yields a provably good approximation ratio. One standard approach is to relax the binary variables x_j to unit vectors v_j in some

Hilbert space $\mathscr{H}$, and to replace the product $x_i x_j$ by the inner product $v_i \cdot v_j$ in $\mathscr{H}$. This gives the following mathematical program:

$$\begin{aligned} \text{maximize} \quad & \sum_{i,j} Q_{ij}(1 - v_i \cdot v_j) \\ \text{subject to} \quad & \|v_k\| = 1, \quad k = 1, 2, \dots, n \end{aligned} \tag{8.2}$$

Problem (8.2) is an instance of an SDP and is a so-called *SDP relaxation* of problem (8.1). Now, let w_{QP} and w_{SDP} be the optimal values of problems (8.1) and (8.2), respectively. It is clear that $w_{SDP} \geq w_{QP}$, since any feasible solution to problem (8.1) is also a feasible solution to problem (8.2). Let $\{v_1^*, \dots, v_n^*\}$ be a solution to problem (8.2), which can be obtained in polynomial time (see, e.g., Refs. [3,4]). Goemans and Williamson [1] then proposed to round this solution via a *random hyperplane*. Specifically, let $r \in \mathbb{R}^n$ be a random vector drawn uniformly from the unit sphere S^{n-1} (see, e.g., Ref. [10] for how this could be done). Then, set $\hat{x}_j = \text{sgn}(v_j^* \cdot r)$, where

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

It is clear that the rounded solution $\{\hat{x}_1, \dots, \hat{x}_n\}$ is a feasible solution to problem (8.1), but how does it compare with the optimal solution? In the case where the entries of Q are nonnegative (i.e., $Q_{ij} \geq 0$ for all i, j), Goemans and Williamson [1] gave the following elegant analysis. First, using a geometric argument and some analysis, one can show that

$$\mathbb{E}\left[1 - \hat{x}_i \hat{x}_j\right] = \frac{2}{\pi} \arccos(v_i^* \cdot v_j^*) \geq c(1 - v_i^* \cdot v_j^*) \tag{8.3}$$

for some constant $c > 0$. Now, since $Q_{ij} \geq 0$ and $1 - v_i^* \cdot v_j^* \geq 0$, we have

$$Q_{ij}\mathbb{E}\left[1 - \hat{x}_i \hat{x}_j\right] \geq c \cdot Q_{ij}(1 - v_i^* \cdot v_j^*) \tag{8.4}$$

Thus, upon summing over i, j, we conclude that

$$\sum_{i,j} Q_{ij}\mathbb{E}\left[1 - \hat{x}_i \hat{x}_j\right] \geq c \sum_{i,j} Q_{ij}(1 - v_i^* \cdot v_j^*)$$

Notice that the right-hand side is simply $c \cdot w_{SDP}$, which is at least $c \cdot w_{QP}$. Thus, it follows that the above algorithm gives an $1/c$-approximation to the optimal value of problem (8.1) in expectation.

Now, consider the following related problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{i,j} Q_{ij} x_i x_j \\ \text{subject to} \quad & x_k \in \{-1, 1\}, \quad k = 1, 2, \dots, n \end{aligned} \tag{8.5}$$

and its natural SDP relaxation:

$$\begin{aligned} \text{maximize} \quad & \sum_{i,j} Q_{ij}(v_i \cdot v_j) \\ \text{subject to} \quad & \|v_k\| = 1, \quad k = 1, 2, \dots, n \end{aligned} \tag{8.6}$$

It is tempting to analyze problem (8.6) using the same approach. Indeed, by using the same rounding scheme, one can show that

$$\mathbb{E}\left[\hat{x}_i \hat{x}_j\right] = \frac{2}{\pi} \arcsin(v_i^* \cdot v_j^*)$$

and that for $-1 \leq t \leq 1$, $\arcsin(x)$ and x differ only by a constant factor. However, as one readily observes, the inequality (8.3) only provides a term-by-term estimate of the objective function and not a global estimate. Thus, if we do not assume that the entries of Q are nonnegative, then the same analysis will not go through, as inequality (8.4) will no longer be valid. However, the bottleneck in the analysis lies in (8.3), where we replace the equality by an inequality. Thus, if we could express $\mathbb{E}[\hat{x}_i\hat{x}_j]$ in such a way so that equality can be preserved throughout, then we may be able to circumvent the aforementioned diffculty and establish approximation guarantees for problem (8.6).

It turns out that such an expression is possible. In his proof of Grothendieck's inequality—a well-known inequality in functional analysis—Rietz [11] has established the following identity:

$$\begin{aligned}\frac{\pi}{2}\mathbb{E}[\operatorname{sgn}(b\cdot G)\ \operatorname{sgn}(c\cdot G)] = (b\cdot c) + \mathbb{E}\Bigg[\Bigg(b\cdot G-\sqrt{\frac{\pi}{2}}\ \operatorname{sgn}(b\cdot G)\Bigg)\\ \times\Bigg(c\cdot G-\sqrt{\frac{\pi}{2}}\ \operatorname{sgn}(c\cdot G)\Bigg)\Bigg]\end{aligned} \tag{8.7}$$

where $b, c \in \mathbb{R}^n$ are unit vectors and $G = (g_1, \ldots, g_n)$ is a standard Gaussian random vector, i.e., the g_i's are i.i.d. standard normal random variables. This identity was established in 1974, but its use for analyzing SDP relaxations was not discovered until 2004, when Alon and Naor [12] used it to analyze the SDP relaxation of a certain quadratic program. To see how this identity can be used to analyze problem (8.6), we first let $G \in \mathbb{R}^n$ be a standard Gaussian random vector and set $\hat{x}_i = \operatorname{sgn}(v_i^*\cdot G)$. Then, using (8.7), we see that

$$\begin{aligned}\frac{\pi}{2}\sum_{i,j}Q_{ij}\mathbb{E}\big[\hat{x}_i\hat{x}_j\big] &= \sum_{i,j}Q_{ij}(v_i^*\cdot v_j^*)\\ &\quad+\sum_{i,j}Q_{ij}\mathbb{E}\Bigg[\Bigg(v_i^*\cdot G-\sqrt{\frac{\pi}{2}}\ \operatorname{sgn}(v_i^*\cdot G)\Bigg)\Bigg(v_j^*\cdot G-\sqrt{\frac{\pi}{2}}\ \operatorname{sgn}(v_j^*\cdot G)\Bigg)\Bigg]\\ &= w_{SDP}+\sum_{i,j}Q_{ij}\mathbb{E}\Bigg[\Bigg(v_i^*\cdot G-\sqrt{\frac{\pi}{2}}\ \operatorname{sgn}(v_i^*\cdot G)\Bigg)\\ &\quad\times\Bigg(v_j^*\cdot G-\sqrt{\frac{\pi}{2}}\ \operatorname{sgn}(v_j^*\cdot G)\Bigg)\Bigg]\end{aligned}$$

We now claim that

$$\sum_{i,j}Q_{ij}\mathbb{E}\Bigg[\Bigg(v_i^*\cdot G-\sqrt{\frac{\pi}{2}}\ \operatorname{sgn}(v_i^*\cdot G)\Bigg)\Bigg(v_j^*\cdot G-\sqrt{\frac{\pi}{2}}\ \operatorname{sgn}(v_j^*\cdot G)\Bigg)\Bigg] \geq 0 \tag{8.8}$$

Assuming this, we see that

$$\sum_{i,j}Q_{ij}\mathbb{E}\big[\hat{x}_i\hat{x}_j\big] \geq \frac{2}{\pi}w_{SDP}$$

thus showing that the above algorithm gives an $2/\pi$-approximation. We remark that Nesterov [8] has established the above result using a different technique, but as we shall see, the technique we presented can be applied to analyze other SDP relaxations as well.

To establish (8.8), let N be the standard Gaussian measure, i.e.,

$$dN(r) = \frac{1}{(2\pi)^{n/2}}\exp\left(-\|r\|^2/2\right)dr$$

where $\|r\|^2 = r_1^2 + \cdots + r_n^2$ and dr is the n-dimensional Lebesgue measure. Consider the Hilbert space $L^2(N)$, i.e., the space of all real-valued measurable functions f on $\mathbb{R}^n$ with $\int_{R^n}|f|^2\,dN < \infty$ (see, e.g., Ref. [13] for details). Recall that the inner product on $L^2(N)$ is given by

$$\langle f_u, f_v\rangle \equiv \int_{R^n} f_u(r)f_v(r)\,dN(r) = \mathbb{E}[f_u\overline{f_v}]$$

Now, observe that for each vector $u \in \mathbb{R}^n$, the function $h_u : \mathbb{R}^n \to \mathbb{R}$ given by

$$h_u(r) = u \cdot r - \sqrt{\frac{\pi}{2}}\, \operatorname{sgn}(u \cdot r)$$

is an element of $L^2(N)$. Thus, it follows that

$$\mathbb{E}\left[h_{v_i^*} h_{v_j^*}\right] = \mathbb{E}\left[\left(v_i^* \cdot G - \sqrt{\frac{\pi}{2}}\, \operatorname{sgn}(v_i^* \cdot G)\right)\left(v_j^* \cdot G - \sqrt{\frac{\pi}{2}}\, \operatorname{sgn}(v_j^* \cdot G)\right)\right]$$

is an inner product of two vectors in the Hilbert space $L^2(N)$. Moreover, we may consider Q as a positive-semidefinite operator defined on the n-dimensional subspace spanned by the vectors $\{h_{v_1^*}, \ldots, h_{v_n^*}\}$. These observations allow us to conclude that (8.8) holds.

It is now instructive to review what we have done. We begin with the identity (8.7), which can be written in the form

$$\gamma \mathbb{E}[f(b \cdot G) f(c \cdot G)] = (b \cdot c) + \mathbb{E}[(b \cdot G - \sqrt{\gamma} f(b \cdot G))(c \cdot G - \sqrt{\gamma} f(c \cdot G))]$$

where f is a rotational invariant rounding function and $\gamma > 0$ a constant. This suggests that by choosing different f's, we may be able to analyze various SDP relaxations. Indeed, this is the idea behind the results in Ref. [14], where the authors showed how to choose appropriate f's to analyze the SDP relaxations of a class of discrete and continuous quadratic optimization problems in complex Hermitian form. Specifically, consider the following problems:

$$\begin{aligned} &\text{maximize} && z^H Q z \\ &\text{subject to} && z_j \in \{1, \omega, \ldots, \omega^{k-1}\}, \quad j = 1, 2, \ldots, n \end{aligned} \tag{8.9}$$

and

$$\begin{aligned} &\text{maximize} && z^H Q z \\ &\text{subject to} && |z_j| = 1, \quad j = 1, 2, \ldots, n \\ &&& z \in C^n \end{aligned} \tag{8.10}$$

where $Q \in C^{n \times n}$ is a Hermitian matrix, ω the principal kth root of unity, and z^H denotes the conjugate transpose of the complex vector $z \in C^n$. The difference between problems (8.9) and (8.10) lies in the values that the decision variables are allowed to take. In problem (8.9), we have discrete decision variables, and such variables can be conveniently modeled as roots of unity. However, in problem (8.10), the decision variables are constrained to lie on the unit circle, which is a continuous domain. Such problems arise from many applications. For instance, the MAX-3-CUT problem where the Laplacian matrix is positive, semidefinite can be formulated as an instance of problem (8.9). On the other hand, problem (8.10) arises from the study of robust optimization as well as control theory [15,16].

Just like their real counterparts, both of these problems are NP-hard, and thus we will settle for approximation algorithms. In the following sections, we will present a generic algorithm and a unified treatment of the two seemingly very different problems (8.9) and (8.10) using their natural SDP relaxations, and to derive approximation guarantees using variants of the identity (8.7).

8.2 Complex Quadratic Optimization

Let $Q \in C^{n \times n}$ be a Hermitian matrix, where $n \geq 1$ is an integer. Consider the following discrete quadratic optimization problem:

$$\begin{aligned} &\text{maximize} && z^H Q z \\ &\text{subject to} && z_j \in \{1, \omega, \ldots, \omega^{k-1}\}, \quad j = 1, 2, \ldots, n \end{aligned} \tag{8.11}$$

where ω is the principal kth root of unity. We note that as k goes to infinity, the discrete problem (8.11) becomes a continuous optimization problem:

$$\begin{aligned} &\text{maximize} && z^H Qz \\ &\text{subject to} && |z_j| = 1, \quad j = 1, 2, \ldots, n \\ & && z \in C^n \end{aligned} \tag{8.12}$$

Although problems (8.11) and (8.12) are quite different in nature, the following *complex* semidefinite program provides a relaxation for both of them:

$$\begin{aligned} &\text{maximize} && Q \bullet Z \\ &\text{subject to} && Z_{jj} = 1, \quad j = 1, 2, \ldots, n \\ & && Z \succeq 0 \end{aligned} \tag{8.13}$$

As before, we use w_{SDP} to denote the optimal value of the SDP relaxation (8.13).

Our goal is to get a near-optimal solution for problems (8.11) and (8.12). Below we present a generic algorithm due to Ref. [14] that can be used to solve both problems (8.11) and (8.12). The algorithm is quite simple, and it is similar in spirit to the algorithm of Goemans and Williamson [1,7].

Algorithm

STEP 1: Solve the SDP relaxation (8.13) and obtain an optimal solution Z^*. Since Z^* is positive-semidefinite, we can obtain a Cholesky decomposition $Z^* = VV^H$, where $V = (v_1, v_2, \ldots, v_n)$.

STEP 2: Generate two independent normally distributed random vectors $x \in R^n$ and $y \in R^n$ with mean 0 and covariance matrix $\frac{1}{2}I_n$, where I_n is the $n \times n$ identity matrix. Let $r = x + yi$.

STEP 3: For $j = 1, 2, \ldots, n$, let $\hat{z}_j = f(v_j \cdot r)$, where the function $f(\cdot)$ depends on the structure of the problem and will be fixed later. Let $\hat{z} = (\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_n)$ be the resulting solution.

To prove the performance guarantee of the above algorithm, we are interested in analyzing the quantity:

$$\mathbb{E}\left[\hat{z}^H Q\hat{z}\right] = \sum_{l,m} Q_{lm}\mathbb{E}[f(v_l \cdot r)\overline{f(v_m \cdot r)}]$$

Thus, it would be sufficient to compute the quantity $\mathbb{E}[f(v_l \cdot r)\overline{f(v_m \cdot r)}]$ for any l, m, and this will be the main concern of the analysis. The analysis, of course, depends on the choice of the function $f(\cdot)$. However, the following Lemma will be useful and it is independent of the function $f(\cdot)$. Recall that for two vectors $b, c \in C^n$, we have $b \cdot c = \sum_{j=1}^n b_j\overline{c_j}$.

Lemma 8.1

For any pair of vectors $b, c \in C^n$, $\mathbb{E}[(b \cdot r)\overline{(c \cdot r)}] = b \cdot c$, where $r = x + yi$ and $x \in R^n$ and $y \in R^n$ are two independent normally distributed random vector with mean 0 and covariance matrix $\frac{1}{2}I_n$.

Proof

This follows from a straightforward computation

$$\mathbb{E}[(b \cdot r)\overline{(c \cdot r)}] = \mathbb{E}\left[\left(\sum_{j=1}^n b_j\overline{r}_j\right)\left(\sum_{k=1}^n \overline{c}_k r_k\right)\right] = \sum_{j,k=1}^n b_j\overline{c}_k\mathbb{E}[\overline{r}_j r_k] = \sum_{j=1}^n b_j\overline{c}_j$$

where the last equality follows from the fact that the entries of x and y are independent, normally distributed with mean 0 and variance 1/2. □

In the sequel, we shall use $r \sim \mathcal{N}_C(0, I_n)$ to indicate that r is an n-dimensional standard complex normal random vector, i.e., $r = x + yi$, where $x, y \in R^n$ are two independent normally distributed random vectors, each with mean 0 and covariance matrix $\frac{1}{2}I_n$.

8.3 Discrete Problems Where Q Is Positive Semidefinite

In this section, we assume that Q is Hermitian and positive semidefinite. Consider the discrete complex quadratic optimization problem (8.11). In this case, we define the function $f(\cdot)$ in the generic algorithm presented in Section 8.2 as follows:

$$f(z) = \begin{cases} 1 & \text{if } \arg(z) \in [-\pi/k, \pi/k) \\ \omega & \text{if } \arg(z) \in [\pi/k, 3\pi/k) \\ \vdots & \vdots \\ \omega^{k-1} & \text{if } \arg(z) \in [(2k-3)\pi/k, (2k-1)\pi/k) \end{cases} \tag{8.14}$$

By construction, we have $\hat{z}_j \in \{1, \omega, \ldots, \omega^{k-1}\}$ for $j = 1, 2, \ldots, n$, i.e., $\hat{z}$ is a feasible solution of problem (8.11). Now, we can establish the following lemma.

Lemma 8.2

For any pair of vectors $b, c \in C^n$ and $r \sim \mathcal{N}_C(0, I_n)$, we have

$$\mathbb{E}[(b \cdot r)\overline{f(c \cdot r)}] = \frac{k \sin(\pi/k)}{2\sqrt{\pi}}(b \cdot c)$$

Proof

By rotation invariance, we may assume without loss of generality that $b = (b_1, b_2, 0, \ldots, 0)$ and $c = (1, 0, \ldots, 0)$. Then, we have

$$\begin{aligned} \mathbb{E}[(b_1\overline{r_1} + b_2\overline{r_2})\overline{f(\overline{r_1})}] &= b_1\mathbb{E}[\overline{r_1}\,\overline{f(\overline{r_1})}] \\ &= \frac{b_1}{\pi}\int_{\mathbf{R}}\int_{\mathbf{R}}(x - iy)\overline{f(x - iy)}\exp\{-(x^2 + y^2)\}\,dx\,dy \\ &= \frac{b_1}{\pi}\int_0^{\infty}\int_0^{2\pi} \rho^2 e^{-i\theta}\overline{f(\rho e^{-i\theta})}e^{-\rho^2}\,d\theta\,d\rho \end{aligned}$$

Now, for any $j = 1, \ldots, k$, if $(2j-3)\pi/k < \theta \le (2j-1)\pi/k$, then $-(2j-1)\pi/k \le -\theta < -(2j-3)\pi/k$, or

$$\frac{2k - 2j + 1}{k}\pi \le 2\pi - \theta < \frac{2k - 2j + 3}{k}\pi$$

It then follows from the definition of $f(\cdot)$ that

$$f(\rho e^{-i\theta}) = f(\rho e^{i(2\pi-\theta)}) = \omega^{k-j+1}$$

and hence $\overline{f(\rho e^{-i\theta})} = \omega^{j-1}$. Therefore, we have

$$\int_{(2j-3)\pi/k}^{(2j-1)\pi/k} \overline{f(\rho e^{-i\theta})}e^{-i\theta}\,d\theta = \omega^{j-1}\int_{(2j-3)\pi/k}^{(2j-1)\pi/k} e^{-i\theta}\,d\theta = 2\sin(\pi/k)$$

In particular, the above quantity is independent of j. Thus, we conclude that

$$\int_0^{2\pi} \overline{f(\rho e^{-i\theta})}e^{-i\theta}\,d\theta = 2k\sin(\pi/k)$$

Moreover, since we have

$$\int_0^{\infty} \rho^2 e^{-\rho^2}\,d\rho = \frac{\sqrt{\pi}}{4}$$

it follows that

$$\mathbb{E}[(b_1\overline{r_1} + b_2\overline{r_2})\overline{f(\overline{r_1})}] = \frac{k\sin(\pi/k)}{2\sqrt{\pi}} b_1 = \frac{k\sin(\pi/k)}{2\sqrt{\pi}}(b \cdot c)$$

as desired. □

We are now ready to prove the main result of this section.

Theorem 8.1

Suppose that Q is Hermitian and positive semidefinite. Then, there exists a $\frac{(k\sin(\frac{\pi}{k}))^2}{4\pi}$ approximation algorithm for problem (8.11).

Proof

By Lemmas 8.1 and 8.2, we have

$$\begin{aligned}\mathbb{E}&\left[\left\{(b \cdot r) - \frac{2\sqrt{\pi}}{k\sin(\frac{\pi}{k})} f(b \cdot r)\right\}\left\{\overline{(c \cdot r) - \frac{2\sqrt{\pi}}{k\sin(\frac{\pi}{k})} f(c \cdot r)}\right\}\right] \\ &= -(b \cdot c) + \frac{4\pi}{(k\sin(\frac{\pi}{k}))^2}\mathbb{E}[f(b \cdot r)\overline{f(c \cdot r)}]\end{aligned}$$

It follows that

$$\begin{aligned}\mathbb{E}[\hat{z}^H Q \hat{z}] = \frac{(k\sin(\frac{\pi}{k}))^2}{4\pi} \sum_{l=1}^{n}\sum_{m=1}^{n} q_{lm}(v_l \cdot v_m) + \frac{(k\sin(\frac{\pi}{k}))^2}{4\pi} \sum_{l=1}^{n}\sum_{m=1}^{n} q_{lm} \\ \times \mathbb{E}\left[\left\{(v_l \cdot r) - \frac{2\sqrt{\pi}}{k\sin(\frac{\pi}{k})} f(v_l \cdot r)\right\}\left\{\overline{(v_m \cdot r) - \frac{2\sqrt{\pi}}{k\sin(\frac{\pi}{k})} f(v_m \cdot r)}\right\}\right]\end{aligned} \tag{8.15}$$

We now claim that

$$\sum_{l=1}^{n}\sum_{m=1}^{n} q_{lm}\mathbb{E}\left[\left\{(v_l \cdot r) - \frac{2\sqrt{\pi}}{k\sin(\frac{\pi}{k})} f(v_l \cdot r)\right\}\left\{\overline{(v_m \cdot r) - \frac{2\sqrt{\pi}}{k\sin(\frac{\pi}{k})} f(v_m \cdot r)}\right\}\right] \geq 0 \tag{8.16}$$

To see this, let G be the standard complex Gaussian measure, i.e.,

$$dG(r) = \frac{1}{\pi^n}\exp\left(-\|r\|^2\right) dr$$

where $\|r\|^2 = |r_1|^2 + \cdots + |r_n|^2$ and dr is the $2n$-dimensional Lebesgue measure. Consider the Hilbert space $L^2(G)$, i.e., the space of all complex-valued measurable functions f on C^n with $\int_{C^n} |f|^2\, dG < \infty$. Recall that the inner product on $L^2(G)$ is given by

$$\langle f_u, f_v \rangle \equiv \int_{C^n} f_u(r)\overline{f_v(r)}\, dG(r) = \mathbb{E}[f_u\overline{f_v}]$$

Now, observe that for each vector $u \in C^n$, the function $h_u : C^n \to C$ given by

$$h_u(r) = u \cdot r - \frac{2\sqrt{\pi}}{k\sin(\frac{\pi}{k})} f(u \cdot r)$$

is an element of $L^2(G)$. Thus, it follows that

$$\mathbb{E}\left[h_{v_l}\overline{h_{v_m}}\right] = \mathbb{E}\left[\left\{(v_l \cdot r) - \frac{2\sqrt{\pi}}{k\sin(\frac{\pi}{k})} f(v_l \cdot r)\right\}\left\{\overline{(v_m \cdot r) - \frac{2\sqrt{\pi}}{k\sin(\frac{\pi}{k})} f(v_m \cdot r)}\right\}\right]$$

is an inner product of two vectors in the Hilbert space $L^2(G)$. Moreover, we may consider Q as a positive semidefinite operator defined on the n-dimensional subspace spanned by the vectors $\{h_{v_1}, \ldots, h_{v_n}\}$.

These observations allow us to conclude that (8.16) holds. Finally, upon substituting (8.16) into (8.15), we obtain

$$\mathbb{E}[\hat{z}^H Q\hat{z}] \geq \frac{(k\sin(\frac{\pi}{k}))^2}{4\pi}\sum_{l=1}^{n}\sum_{m=1}^{n} q_{lm}(v_l \cdot v_m) = \frac{(k\sin(\frac{\pi}{k}))^2}{4\pi} w_{SDP}$$

i.e., our algorithm gives a $\frac{(k\sin(\frac{\pi}{k}))^2}{4\pi}$-approximation. □

It is interesting to note that the above result can be obtained via a completely different technique. In Ref. [17], the authors developed a closed-form formula for computing the probability of a complex-valued normally distributed bivariate random vector to be in a given angular region. Using this formula and the series expansions of certain trigonometric functions, they are able to establish the same result.

As an application of Theorem 8.1, we consider the MAX-3-CUT problem, which is defined as follows. We are given an undirected graph $G = (V, E)$ with V being the set of nodes and E being the set of edges. For each edge $(i, j) \in E$, there is a weight w_{ij} that could be positive or negative. For a partition of V into three subsets V_1, V_2, and V_3, we define

$$\delta(V_1, V_2, V_3) = \{(i, j) \in E : i \in V_k,\ j \in V_l \quad \text{for } k \neq l\}$$

and

$$w(\delta(V_1, V_2, V_3)) = \sum_{(i,j)\in\delta(V_1,V_2,V_3)} w_{ij}$$

Our goal is to find a tripartition (V_1, V_2, V_3) such that $w(\delta(V_1, V_2, V_3))$ is maximized. Note that the MAX-3-CUT problem is a generalization of the well-known MAX–CUT problem. In the MAX–CUT problem, we require one of the subsets, say V_3, to be empty.

Goemans and Williamson [7] have given the following complex QP formulation for the MAX-3-CUT problem:

$$\begin{array}{ll} \text{maximize} & \frac{1}{3}\sum_{(i,j)\in E} w_{ij}\left(2 - z_i \cdot z_j - z_j \cdot z_i\right) \\ \text{subject to} & z_j \in \{1, \omega, \omega^2\} \ \text{ for all } \ j \in V \end{array} \tag{8.17}$$

Based on this formulation and its SDP relaxation, Goemans and Williamson [7] were able to give an 0.836-approximation algorithm for the MAX-3-CUT problem when the weights of the edges are nonnegative, i.e., $w_{ij} \geq 0$ for all $(i, j) \in E$. (They also showed that their algorithm is actually the same as that of Frieze and Jerrum [5], and thus give a tighter analysis of the algorithm in Ref. [5].) However, their analysis does not apply if some of the edges have negative weights.

Note that since $w_{ij} = w_{ji}$, problem (8.17) is equivalent to

$$\begin{array}{ll} \text{maximize} & \frac{2}{3}z^H L z \\ \text{subject to} & z_j \in \{1, \omega, \omega^2\} \ \text{ for all } \ j \in V \end{array} \tag{8.18}$$

where L is the Laplacian matrix of the graph $G = (V, E)$, i.e., $L_{ij} = -w_{ij}$ and $L_{ii} = \sum_{j:(i,j)\in E} w_{ij}$. However, by Theorem 8.1, problem (8.18) can be approximated by a factor of $\frac{(3\sin(\frac{\pi}{3}))^2}{4\pi} \approx 0.537$. Therefore, we obtain the following result:

Corollary 8.1

There is a randomized 0.537*-approximation algorithm for the* MAX-3-CUT *problem when the Laplacian matrix is positive-semidefinite.*

8.4 Continuous Problems Where Q Is Positive-Semidefinite

Now, let us consider problem (8.12) when Q is positive-semidefinite. This problem can be seen as a special case of problem (8.11) by letting $k \to \infty$. In this case, the function $f(\cdot)$ is defined as follows:

$$f(t) = \begin{cases} \frac{t}{|t|} & \text{if } |t| > 0 \\ 0 & \text{if } t = 0 \end{cases} \tag{8.19}$$

Note that as $k \to \infty$, we have $\frac{(k \sin(\frac{\pi}{k}))^2}{4\pi} \to \pi/4$. This establishes the following result, which has been proved independently by Ben-Tal et al. [16] and Zhang and Huang [17]. However, the proof presented above is quite a bit simpler.

Corollary 8.2

Suppose that Q is positive semidefinite and Hermitian. Then, there exists a $\frac{\pi}{4}$-approximation algorithm for problem (8.12).

8.5 Continuous Problems Where Q Is Not Positive-Semidefinite

In this section, we deal with problem (8.12) where the matrix Q is not positive-semidefinite. However, for convenience, we assume that $w_{SDP} > 0$ so that the standard definition of approximation algorithm makes sense for our problem. It is clear that $w_{SDP} > 0$ as long as all the diagonal entries of Q are zeros.

Assumption 8.1

The diagonal entries of Q are all zeros, i.e., $Q_{ii} = 0$ for $i = 1, 2, \ldots, n$.

In fact, Assumption 8.1 leads to the even stronger result that follows.

Lemma 8.3

If Q satisfies Assumption 8.1, then there exists a constant $C > 0$ such that

$$w_{SDP} \geq C \sqrt{\sum_{1 \leq i, j \leq n} |q_{ij}|^2} > 0$$

Proof

It is straightforward to show that problem (8.12) is equivalent to

$$\begin{aligned} &\text{maximize} && (x^T, y^T) \begin{pmatrix} Re(Q) & Im(Q) \\ -Im(Q) & Re(Q) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\ &\text{subject to} && x_j^2 + y_j^2 = 1, \quad j = 1, 2, \ldots, n \\ &&& x, y \in R^n \end{aligned} \tag{8.20}$$

Moreover, the objective value of problem (8.20) is bounded below by the objective value of the following problem:

$$\begin{aligned} &\text{maximize} && (x^T, y^T) \begin{pmatrix} Re(Q) & Im(Q) \\ -Im(Q) & Re(Q) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\ &\text{subject to} && x_j^2 = \tfrac{1}{2}, \quad j = 1, 2, \ldots, n \\ &&& y_j^2 = \tfrac{1}{2}, \quad j = 1, 2, \ldots, n \\ &&& x, y \in R^n \end{aligned} \tag{8.21}$$

Since Q satisfies Assumption 8.1, the diagonal entries of

$$\begin{pmatrix} Re(Q) & Im(Q) \\ -Im(Q) & Re(Q) \end{pmatrix}$$

must also be zeros. It has been shown in Ref. [18] that for any real matrix $A = (a_{ij})_{n\times n}$ with a zero diagonal, the optimal objective value of

$$\begin{aligned} &\text{maximize} && x^T Ax \\ &\text{subject to} && x_j^2 = 1, \quad j = 1, 2, \ldots, n \\ & && x \in R^n \end{aligned} \tag{8.22}$$

is bounded below by $C\sqrt{2\sum_{1\le i,\, j\le n} |a_{ij}|^2}$, for some constant $C > 0$ which is independent of A. This implies that the optimal objective value of problem (8.21) is at least

$$\frac{1}{2}C\sqrt{2\sum_{1\le i,\, j\le n} 2(|Re(q_{ij})|^2 + |Im(q_{ij})|^2)} \ge C\sqrt{\sum_{1\le i,\, j\le n} |q_{ij}|^2}$$

which leads to the desired result. □

Again, we use our generic algorithm presented in Section 8.2. In this case, we specify the function $f(\cdot)$ as follows:

$$f(t) = \begin{cases} \frac{t}{T} & \text{if } |t| \le T \\ \frac{t}{|t|} & \text{if } |t| > T \end{cases} \tag{8.23}$$

where T is a parameter which will be fixed later. If we let $z_j = f(v_j \cdot r)$, then the solution $z = (z_1, \ldots, z_n)$ obtained by this rounding may not be feasible, as the point may not have unit modulus. However, we know that $|z_j| \le 1$. Thus, we can further round the solution as follows:

$$\hat{z} = \begin{cases} z/|z| & \text{with probability } (1 + |z|)/2 \\ -\bar{z}/|z| & \text{with probability } (1 - |z|)/2 \end{cases}$$

The following lemma is a direct consequence of the second randomized rounding.

Lemma 8.4

For $i \ne j$, we have $\mathbb{E}[\hat{z}_i\overline{\hat{z}_j}] = \mathbb{E}[z_i\overline{z_j}]$.

Proof

By definition, conditioning on z_i, z_j, we have

$$\begin{aligned} \mathbb{E}[\hat{z}_i\overline{\hat{z}_j} \mid z_i, z_j] &= \Pr\{\hat{z}_i = z_i/|z_i|,\ \hat{z}_j = z_j/|z_j|\}\frac{z_i\overline{z_j}}{|z_i|\cdot|\overline{z_j}|} \\ &\quad + \Pr\{\hat{z}_i = z_i/|z_i|,\ \hat{z}_j = -z_j/|z_j|\} - \frac{z_i\overline{z_j}}{|z_i|\cdot|\overline{z_j}|} \\ &\quad + \Pr\{\hat{z}_i = -z_i/|z_i|,\ \hat{z}_j = z_j/|z_j|\} - \frac{z_i\overline{z_j}}{|z_i|\cdot|\overline{z_j}|} \\ &\quad + \Pr\{\hat{z}_i = -z_i/|z_i|,\ \hat{z}_j = -z_j/|z_j|\}\frac{z_i\overline{z_j}}{|z_i|\cdot|\overline{z_j}|} \\ &= \frac{1}{2}(1 + |z_i|\cdot|z_j|)\frac{z_i\overline{z_j}}{|z_i|\cdot|\overline{z_j}|} - \frac{1}{2}(1 - |z_i|\cdot|z_j|)\frac{z_i\overline{z_j}}{|z_i|\cdot|\overline{z_j}|} \\ &= z_i\overline{z_j} \end{aligned}$$

The desired result then follows from the tower property of conditional expectation. □

This shows that the expected value of the solution on the circle equals that of the "fractional" solution obtained by applying $f(\cdot)$ to the SDP solution. Therefore, we could still restrict ourselves to the rounding function $f(\cdot)$.

Now, define

$$g(T) = \frac{1}{T} - \frac{1}{T}e^{-T^2} + \sqrt{\pi}(1 - \Phi(\sqrt{2}T))$$

where $\Phi(\cdot)$ is the probability distribution function of $\mathcal{N}(0, 1)$.

Lemma 8.5

For any pair of vectors $b, c \in C^n$ and $r \sim \mathcal{N}_C(0, I_n)$, we have

$$\mathbb{E}[(b \cdot r)\overline{f(c \cdot r)}] = g(T)(b \cdot c)$$

Proof

Again, without loss of generality, we assume that $c = (1, 0, \ldots, 0)$ and $b = (b_1, b_2, 0, \ldots, 0)$. Let $\mathbf{1}_A$ be the indicator function of the set A, i.e., $\mathbf{1}_A(\omega) = 1$ if $\omega \in A$ and $\mathbf{1}_A(\omega) = 0$ otherwise. Then, we have

$$\begin{aligned}
\mathbb{E}[(b \cdot r)\overline{f(c \cdot r)}] &= \mathbb{E}\left[(b_1\bar{r}_1 + b_2\bar{r}_2)\frac{r_1}{T} \cdot \mathbf{1}_{\{|r_1| \le T\}}\right] + \mathbb{E}\left[(b_1\bar{r}_1 + b_2\bar{r}_2)\frac{r_1}{|r_1|} \cdot \mathbf{1}_{\{|r_1| > T\}}\right] \\
&= \frac{1}{T}\mathbb{E}\left[b_1|r_1|^2 \cdot \mathbf{1}_{\{|r_1| \le T\}}\right] + \mathbb{E}\left[b_1|r_1| \cdot \mathbf{1}_{\{|r_1| > T\}}\right] \\
&= \frac{b_1}{T} \cdot \frac{1}{\pi}\int_{x^2+y^2 \le T^2} (x^2 + y^2)\ \exp(-(x^2 + y^2))\, dx\, dy \\
&\quad + \frac{b_1}{\pi}\int_{x^2+y^2 > T^2} \sqrt{x^2 + y^2}\ \exp(-(x^2 + y^2))\, dx\, dy \\
&= \frac{b_1}{\pi T}\int_0^{2\pi}\int_0^T \rho^3 \exp(-\rho^2)\, d\rho\, d\theta + \frac{b_1}{\pi}\int_0^{2\pi}\int_T^\infty \rho^2 \exp(-\rho^2)\, d\rho\, d\theta \\
&= g(T)b_1
\end{aligned}$$

where the last equality follows from the facts

$$\int_0^T \rho^3 \exp(-\rho^2)\, d\rho = \frac{1}{2}\left(1 - (T^2 + 1)\exp(-T^2)\right)$$

and

$$\int_T^\infty \rho^2 \exp(-\rho^2)\, d\rho = \frac{1}{2}\left(T\exp(-T^2) + \sqrt{\pi}(1 - \Phi(\sqrt{2}T))\right)$$

This completes the proof. □

In a similar fashion, one can show the following:

Lemma 8.6

For any pair of vectors $b, c \in C^n$ and $r \sim \mathcal{N}_C(0, I_n)$, we have

$$\mathbb{E}[f(c \cdot r)\overline{f(c \cdot r)}] = \frac{1}{T^2} - \frac{1}{T^2}\exp(-T^2)$$

Now, by putting everything together, we obtain the following:

Theorem 8.2

If Q satisfies Assumption 8.1, then there exists a constant $C > 0$ such that

$$\mathbb{E}\left[\hat{z}^H Q\hat{z}\right] \geq \frac{1}{3\ln(\beta)}\, w_{SDP}$$

where $\beta = \max\left\{5, \frac{\sum_{1\leq k,m\leq n}|q_{km}|}{C\sqrt{\sum_{1\leq k,m\leq n}|q_{km}|^2}}\right\}$

Proof

By Lemmas 8.1 and 8.5, we have

$$\mathbb{E}[\{(b\cdot r) - Tf(b\cdot r)\}\overline{\{(c\cdot r) - Tf(c\cdot r)\}}] = (1-2Tg(T))(b\cdot c) + T^2\mathbb{E}[f(b\cdot r)\overline{f(c\cdot r)}]$$

It follows that

$$\begin{aligned}\mathbb{E}\left[\hat{z}^H Q\hat{z}\right] &= \sum_{k=1}^{n}\sum_{m=1}^{n}\frac{2Tg(T)-1}{T^2}q_{km}(v_k\cdot v_m)\\ &\quad + \frac{1}{T^2}\sum_{k=1}^{n}\sum_{m=1}^{n}q_{km}\mathbb{E}[\{(v_k\cdot r) - Tf(v_k\cdot r)\}\overline{\{(v_m\cdot r) - Tf(v_m\cdot r)\}}]\end{aligned}$$

Again, the quantity $\mathbb{E}[\{(b\cdot r) - Tf(b\cdot r)\}\overline{\{(c\cdot r) - Tf(c\cdot r)\}}]$ can be seen as an inner product of two vectors in a Hilbert space. Moreover, by letting $b = c$ and using Lemma 8.6, we know that the norm of an Euclidean unit vector in this Hilbert space is

$$2 - 2Tg(T) - \exp(-T^2) = \exp(-T^2) - 2T\sqrt{\pi}(1-\Phi(\sqrt{2}T))$$

It follows that

$$\begin{aligned}&\frac{1}{T^2}\sum_{k=1}^{n}\sum_{m=1}^{n}q_{km}\mathbb{E}[\{(v_k\cdot r) - Tf(v_k\cdot r)\}\overline{\{(v_m\cdot r) - Tf(v_m\cdot r)\}}]\\ &\quad\geq -\frac{\exp(-T^2) - 2T\sqrt{\pi}(1-\Phi(\sqrt{2}T))}{T^2}\sum_{k=1}^{n}\sum_{m=1}^{n}|q_{km}|\end{aligned}$$

In contrast, by Lemma 8.3, we have $w_{SDP} \geq C\sqrt{\sum_{1\leq k,m\leq n}|q_{km}|^2} > 0$ for some constant $C > 0$. It follows that

$$\begin{aligned}&\frac{1}{T^2}\sum_{k=1}^{n}\sum_{m=1}^{n}q_{km}\mathbb{E}[\{(v_k\cdot r) - Tf(v_k\cdot r)\}\overline{\{(v_m\cdot r) - Tf(v_m\cdot r)\}}]\\ &\quad\geq -\frac{\exp(-T^2) - 2T\sqrt{\pi}(1-\Phi(\sqrt{2}T))}{T^2}\,\frac{\sum_{1\leq k,m\leq n}|q_{km}|}{C\sqrt{\sum_{1\leq k,m\leq n}|q_{km}|^2}}\cdot w_{SDP}\\ &\quad\geq -\frac{\exp(-T^2) - 2T\sqrt{\pi}(1-\Phi(\sqrt{2}T))}{T^2}\beta\cdot w_{SDP}\end{aligned}$$

where $\beta = \max\left\{5, \frac{\sum_{1\leq k,m\leq n}|q_{km}|}{C\sqrt{\sum_{1\leq k,m\leq n}|q_{km}|^2}}\right\}$. This implies that

$$\begin{aligned}\mathbb{E}\left[\hat{z}^H Q\hat{z}\right] &\geq \left(\frac{2Tg(T)-1}{T^2} - \frac{\exp(-T^2) - 2T\sqrt{\pi}(1-\Phi(\sqrt{2}T))}{T^2}n\right)w_{SDP}\\ &\geq \frac{1-(2+\beta)\exp(-T^2)}{T^2}\,w_{SDP}\end{aligned}$$

By letting $T = \sqrt{2\ln\beta}$, we have $\mathbb{E}\left[\hat{z}^H Q\hat{z}\right] \geq \frac{1}{3\ln\beta} w_{SDP}$. □

Note that

$$\frac{\sum_{1\leq k,m\leq n}|q_{km}|}{\sqrt{\sum_{1\leq k,m\leq n}|q_{km}|^2}} \leq \frac{1}{n}$$

Therefore, we have

Corollary 8.3

If Q satisfies Assumption 8.1, then $\mathbb{E}\left[\hat{z}^H Q\hat{z}\right] \geq O(\frac{1}{\ln n})\ w_{SDP}$.

8.6 Discrete Problems Where *Q* Is Not Positive-Semidefinite

Let us now consider problem (8.11) where Q is an indefinite Hermitian matrix with diag$(Q) = 0$. Its approximability was left open in Ref. [14] and is recently resolved by Huang and Zhang [19]. It is interesting to note that the techniques of Huang and Zhang encompass well-known ideas in the literature. Specifically, their rounding is similar to a technique introduced in Ref. [20], and their analysis uses some of the ideas from Ref. [21]. To be precise, let Z^* be an optimal solution to problem (8.11). Then, let $r \sim \mathcal{N}_C(0, Z^*)$ be a Gaussian random vector, and for each $j = 1, 2, \ldots, n$, set

$$z'_j = \begin{cases} r_j/|r_j| & \text{if } |r_j| > T \\ r_j/T & \text{if } |r_j| \leq T \end{cases}$$

where $T > 0$ is a parameter to be chosen later. As earlier, each z'_j satisfies $|z'_j| \leq 1$. However, the solution $\{z'_1, \ldots, z'_n\}$ is not feasible to problem (8.11). Thus, we need to perform a second randomized rounding as follows:

$$\hat{z}_j = \omega^l \quad \text{with probability } (1 + \text{Re}(\omega^{-l}z'_j))/k$$

where $l = 0, 1, \ldots, k-1$ and $j = 1, 2, \ldots, n$. Observe that

$$\sum_{l=0}^{k-1}\frac{(1+\text{Re}(\omega^{-l}z'_j))}{k} = 1 + \frac{1}{k}\text{Re}\left[\left(\sum_{l=0}^{k-1}\omega^{-l}\right)z'_j\right] = 1$$

and hence, we indeed have a probability distribution. The following lemma is similar in spirit to Lemma 8.4 and is crucial to the analysis.

Lemma 8.7

For $j \neq l$ *and* $k \geq 3$, *we have* $\mathbb{E}\left[\hat{z}_j\overline{\hat{z}_l}\right] = \frac{1}{4}\mathbb{E}\left[z'_j\overline{z'_l}\right]$.

To analyze the quality of the solution $\{\hat{z}_1, \ldots, \hat{z}_n\}$, we need the following lemma (see also Ref. [21]):

Lemma 8.8

For $j \neq l$ *and* $T > 1$, *we have* $\mathbb{E}[|\Delta_{jl}|] < \exp(-T^2)(4 + 5/T)$, *where* $\Delta_{jl} = r_j\overline{r_l}/T^2 - z'_j\overline{z'_l}$.

Proof

We first divide C^2 into the following (possibly overlapping) regions:

$$A = \{(r_j, r_l) : |r_j| \leq T,\ |r_l| \leq T\}, \quad B = \{(r_j, r_l) : |r_l| > T\}, \quad C = \{(r_j, r_l) : |r_j| > T\}$$

By symmetry, we may assume that $\mathbb{E}[|\Delta_{jl}|\cdot \mathbf{1}_C] \le \mathbb{E}[|\Delta_{jl}|\cdot \mathbf{1}_B]$. Thus, we have

$$\mathbb{E}[|\Delta_{jl}|] \le \mathbb{E}[|\Delta_{jl}|\cdot \mathbf{1}_A] + \mathbb{E}[|\Delta_{jl}|\cdot \mathbf{1}_B] + \mathbb{E}[|\Delta_{jl}|\cdot \mathbf{1}_C] \le \mathbb{E}[|\Delta_{jl}|\cdot \mathbf{1}_A] + 2\mathbb{E}[|\Delta_{jl}|\cdot \mathbf{1}_B]$$

By construction, we have $\mathbb{E}[|\Delta_{jl}|\cdot \mathbf{1}_A] = 0$. Now, suppose that $\mathbb{E}[r_j\overline{r_l}] = Z^*_{jl} = \gamma e^{i\alpha}$. Since we have

$$\begin{pmatrix} r_j \\ r_l \end{pmatrix} \sim \mathcal{N}_C\left(0, \begin{pmatrix} 1 & \gamma e^{i\alpha} \\ \gamma e^{-i\alpha} & 1 \end{pmatrix}\right)$$

it follows that

$$r_j = \gamma e^{i\alpha}\eta + \sqrt{1-\gamma^2}\,\lambda, \quad r_l = \eta$$

where

$$\begin{pmatrix} \eta \\ \lambda \end{pmatrix} \sim \mathcal{N}_C(0, I_2).$$

Hence, we conclude that

$$P(|r_l| > T) = P(|\eta| > T) = \frac{1}{\pi}\int_T^\infty \int_0^{2\pi} re^{-r^2}\, d\theta\, dr = 2\int_T^\infty re^{-r^2}\, dr = e^{-T^2}$$

Now, note that $\mathbb{E}[|z'_j\overline{z'_l}|] \le \mathbb{E}[\mathbf{1}_B] = P(|\eta| > T) = e^{-T^2}$. Moreover, since $\gamma \le 1$, we have

$$\begin{aligned}
\mathbb{E}[|r_j\overline{r_l}|\cdot \mathbf{1}_B] &= \mathbb{E}\left[\left(\gamma e^{i\alpha}|\eta|^2 + \sqrt{1-\gamma^2}\,\lambda\bar{\eta}\right)\cdot \mathbf{1}_B\right] \\
&\le \mathbb{E}\left[\left(|\eta|^2 + |\eta|\cdot|\lambda|\right)\cdot \mathbf{1}_B\right] \\
&= \frac{1}{\pi}\int_T^\infty\int_0^{2\pi} r^3e^{-r^2}\, d\theta\, dr + \left(\frac{1}{\pi}\int_T^\infty\int_0^{2\pi} r^2\, d\theta\, dr\right)\left(\frac{1}{\pi}\int_0^\infty\int_0^{2\pi} r^2e^{-r^2}\, d\theta\, dr\right) \\
&= \left(T^2 + \frac{\sqrt{\pi}\,T}{2} + 1\right)e^{-T^2} + \frac{\pi}{2}\left(1 - \Phi(\sqrt{2}T)\right)
\end{aligned}$$

where $\Phi(\cdot)$ is the cumulative distribution function of the real-valued standard normal distribution. It then follows that

$$\mathbb{E}[|\Delta_{jl}|\cdot \mathbf{1}_B] \le \mathbb{E}\left[\frac{|r_j\overline{r_l}|}{T^2} + |z'_j\overline{z'_l}|\right] \le e^{-T^2}\left(\frac{1}{T^2} + \frac{\sqrt{\pi}}{2T} + 2\right) + \frac{\pi}{2T^2}(1 - \Phi(\sqrt{2}T))$$

whence

$$\mathbb{E}[|\Delta_{jl}|] \le e^{-T^2}\left(\frac{2}{T^2} + \frac{\sqrt{\pi}}{T} + 4\right) + \frac{\pi}{T^2}(1 - \Phi(\sqrt{2}T)) \tag{8.24}$$

Now, observe that

$$\frac{\pi}{T^2}(1 - \Phi(\sqrt{2}T)) = \frac{\sqrt{\pi}}{T^2}\int_T^\infty e^{-s^2}\, ds \le \frac{\sqrt{\pi}}{T^2}\int_T^\infty se^{-s^2}\, ds = \frac{\sqrt{\pi}}{2T^2}e^{-T^2}$$

and hence it follows from (8.24) that

$$\mathbb{E}[|\Delta_{jl}|] \le e^{-T^2}\left(\frac{\sqrt{\pi}+4}{2T^2} + \frac{\sqrt{\pi}}{T} + 4\right) < e^{-T^2}\left(4 + \frac{5}{T}\right)$$

as desired. □

Now, by using Lemmas 8.7 and 8.8, and the techniques similar to that in Ref. [21], we are led to the following result:

Theorem 8.3

Suppose that $n \geq 3$, *and set* $T = \sqrt{9 \ln n}$. *Then, we have* $\mathbb{E}[z^H Qz] \geq O\left(\frac{1}{\ln n}\right) w_{SDP}$.

8.7 Summary

We presented a generic algorithm and a unified treatment of the two seemingly very different quadratic optimization problems in complex Hermitian form. Since these problems are NP-hard, we settled for approximation algorithms. We used their natural SDP relaxations, and to derive approximation guarantees we used variants of the Rietz identity [11].

References

[1] Goemans, M. X. and Williamson, D. P., Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *J. ACM,* 42(6), 1115, 1995.

[2] Hochbaum, D. S., Ed., *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, Boston, MA, 1997.

[3] Alizadeh, F., Interior points methods in semidefinite programming with applications to combinatorial optimization, *SIAM J. Optim.,* 5(1), 13, 1995.

[4] Vandenberghe, L. and Boyd, S., Semidefinite programming, *SIAM Rev.,* 38(1), 49, 1996.

[5] Frieze, A. and Jerrum, M., Improved approximation algorithms for max k-cut and max bisection, *Algorithmica*, 18, 67, 1997.

[6] Goemans, M. X. and Rendl, F., Combinatorial optimization, in *Handbook of Semidefinite Programming: Theory, Algorithms and Applications*, Wolkowicz, H., Saigal, R., and Vandenberghe, L. Eds., Kluwer Academic Publishers, Dordrecht, 2000.

[7] Goemans, M. X. and Williamson, D. P., Approximation algorithms for MAX-3-CUT and other problems via complex semidefinite programming, *J. Comput. Syst. Sci.,* 68(2), 442, 2004.

[8] Nesterov, Y., Global Quadratic Optimization via Conic Relaxation, CORE Discussion Paper 9860, Université Catholique de Louvain, 1998.

[9] Ye, Y., Approximating quadratic programming with bound and quadratic constraints, *Math. Program.,* 84, 219, 1999.

[10] Hicks, J. S. and Wheeling, R. F., An efficient method for generating uniformly distributed points on the surface of an n-dimensional sphere, *Commun. ACM,* 2(4), 17, 1959.

[11] Rietz, R. E., A proof of the Grothendieck inequality, *Israel J. Math.,* 19, 271, 1974.

[12] Alon, N. and Naor, A., Approximating the cut-norm via Grothendieck's inequality, *Proc. of the 36th Annual ACM Symposium on Theory of Computing*, 2004.

[13] Young, N., *An Introduction to Hilbert Space*, Cambridge University Press, Cambridge, UK, 1988.

[14] So, A. M.-C., Zhang, J., and Ye, Y., On approximating complex quadratic optimization problems via semidefinite programming relaxations, *Proc. of the 11th Conf. on Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science 3509*, Vol. 125, Springer, Berlin, 2005, pp. 125–135.

[15] Toker, O. and Özbay, H., On the complexity of purely complex μ computation and related problems in multidimensional systems, *IEEE Trans. Autom. Control,* 43(3), 409, 1998.

[16] Ben-Tal, A., Nemirovski, A., and Roos, C., Extended matrix cube theorems with applications to μ-theory in control, *Math. Oper. Res.,* 28(3), 497, 2003.

[17] Zhang, S. and Huang, Y., Complex Quadratic Optimization and Semidefinite Programming, Technical report SEEM2004–03, Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, 2004; to appear in *SIAM J. Optimization.*

[18] Alon, N., Makarychev, K., Makarychev, Y., and Naor, A., Quadratic forms on graphs, *Proc. of the 37th Annual ACM Symposium on Theory of Computing*, 2005.

[19] Huang, Y. and Zhang, S., Approximation Algorithms for Indefinite Complex Quadratic Maximization Problems, Technical report SEEM2005–03, Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, 2005.

[20] Bertsimas, D. and Ye, Y., Semidefinite relaxations, multivariate normal distributions, and order statistics, in *Handbook of Combinatorial Optimization*, Du, D.-Z and Pardalos, P. M., Eds., Vol. 3, Kluwer Academic Publishers, Dordrecht, 1998, p. 1.

[21] Charikar, M. and Wirth, A., Maximizing quadratic programs: extending Grothendieck's inequality, *Proc. of the 45th Annual IEEE Symp. on Foundations of Computer Science*, 2004.

9

Polynomial-Time Approximation Schemes

Hadas Shachnai
Technion—Israel Institute of Technology

Tami Tamir
The Interdisciplinary Center

9.1 Introduction

Let Π be an NP-hard optimization problem, and let A be an approximation algorithm for Π. For an instance I of Π, let $A(I)$ denote the objective value when running A on I, and let $OPT(I)$ denote the optimal objective value. The approximation ratio of A for the instance I is $R_A(I) = A(I)/OPT(I)$, thus, when Π is minimization (maximization) problem $R_A(I) \geq 1$ ($R_A(I) \leq 1$).

A polynomial-time approximation scheme (PTAS) is an algorithm that takes as input an additional parameter, $\varepsilon > 0$, which determines the desired approximation ratio. As ε approaches 0, the approximation ratio gets arbitrarily close to 1. The time complexity of the scheme is polynomial in the input size, but may be exponential in $1/\varepsilon$. This gives a clear trade-off between running time and quality of approximation. Formally,

Definition 9.1

An approximation scheme *for an optimization problem* Π *is an algorithm* A *which takes as input an instance* I *of* Π *and an error bound* ε*, runs in time polynomial in* $|I|$*, and has approximation ratio* $R_A(I, \varepsilon) \leq (1+\varepsilon)$*. In fact, such an algorithm* A *is a family of algorithms* A_ε *such that for any instance* I*,* $R_{A_\varepsilon}(I) \leq (1+\varepsilon)$.

The approximation algorithm A may be deterministic or randomized. In the latter case, the result is a randomized approximation scheme.

Definition 9.2

A randomized approximation scheme *for an optimization problem* Π *is a family of algorithms* A_ε *which run in time polynomial in* $|I|$ *and have, for any instance* I*,* expected *approximation ratio* $EXP[R_{A_\varepsilon}(I)] \leq (1+\varepsilon)$.

In some approximation schemes, an additive constant k, whose value is independent of I and ε, is added to the approximation ratio. Asymptotically, this constant is negligible, thus, such a scheme is called an asymptotic PTAS.

Definition 9.3

An asymptotic approximation scheme *for an optimization problem* Π *is a family of algorithms* A_ε *that run in time polynomial in* $|I|$*, such that, for some constant k and for any instance* I, $A_\varepsilon(I) \leq (1+\varepsilon)OPT(I)+k$.

We refer the reader to Chapter 11 in this handbook for a detailed study of such schemes.

Some approximation algorithms provide a solution for a *relaxed instance* of the problem. For example, in packing problems, an algorithm may pack the items in bins whose sizes are slightly larger than the original. The objective value is achieved relative to the relaxed instance. This type of algorithm is called a *dual* approximation algorithm [1], or approximation with *resource augmentation* [2]. A *dual approximation scheme* is a family of algorithms A_ε that run in time polynomial in $|I|$, such that, for any instance I, $A(I) \leq (1+\varepsilon)OPT(I)$, and $A(I)$ is achieved for resources augmented by factor of $(1+\varepsilon)$.

Depending on the function $f(|I|, 1/\varepsilon)$, which gives the running time of the scheme, some schemes are classified as *quasi-polynomial* and others as *fully polynomial*. In particular, when the running time is $O(n^{polylog(n)})$ we get a quasi-PTAS (see, e.g., Refs. [3,4]); when the running time is polynomial in both $|I|$ and $1/\varepsilon$ we get a fully polynomial-time approximation scheme (FPTAS). Such schemes are studied in detail in Chapter 10.

There is wide literature on approximation schemes for NP-hard problems. Many of these works present PTASs for certain subclasses of instances of problems, which are in general extremely hard to solve. While some of the proposed schemes may have running times which render them inefficient in practice, these works essentially help identify the class of problems that admit PTAS. There have been some studies also toward characterizing this class of problems (see, e.g., Refs. [5,6] and Chapter 17 of this book). We focus here on the techniques that have been repeatedly used in developing PTASs.

We refer the reader also to the comprehensive survey on Approximation Algorithms by Motwani [7], a tutorial by Schuurman and Woeginger [8], and the survey on scheduling by Karger et al. [9], from which we borrowed some of the examples in this chapter.

9.2 Partial Enumeration

9.2.1 Extending Partial Small-Size Solutions

There are two main techniques based on extending partial small-size solutions. The first technique exploits our ability to solve the problem *optimally* on a constant-size subset of the instance. Thus, initially, such a constant-size subset is selected. This subset contains the most "significant" elements in the instance. We identify elements as *significant* depending on the problem at hand. The problem is solved optimally for this subset. This can be done by exhaustive search, since there is only a constant number of elements to consider. Next, this optimal partial solution is extended into a complete one, using some heuristic which has a bounded approximation ratio.

In the second technique, none of the elements is initially identified as "significant"; instead, *all* partial solutions of constant size are considered, and each is extended to a complete solution using some heuristic. The best extension is selected to be the output of the scheme.

The time-complexity analysis of such PTASs is based on the fact that the number of possible subsets, or solutions that are considered, is exponential in the (constant) size of these subsets. The step in which the constant-size partial solution is extended is usually based on some greedy rule that may require sorting, and is polynomial. The parameter ε specifying the required approximation ratio of $(1+\varepsilon)$ determines the size k of the partial solution to which an exponential exhaustive search is applied. This implies that the running time of such schemes is exponential in $1/\varepsilon$.

9.2.1.1 Extending an Optimal Solution for a Single Subset

We present the first technique in the context of a classical scheduling problem, namely, the problem of finding the *minimum makespan* (MM) (or, overall completion time) of a schedule of n jobs on m identical machines. The main idea in the PTAS of Graham [10] is to first optimally schedule the k longest jobs and then schedule, using some heuristic, the remaining jobs. Formally, the input for the MM problem consists of n jobs and m identical machines. The goal is to schedule the jobs nonpreemptively on the machines in a way that minimizes the maximum completion time of any job in the schedule.

Denote by $p_1, \dots, p_n$ the processing times of the jobs. Assume that $n > m$, and that the processing times are sorted in nonincreasing order, that is, for all $i < j$, $p_i \geq p_j$. A well-known heuristic for the makespan problem is the *LPT rule*, which selects the longest unscheduled job in the sorted list and assigns it to a processor which currently has the minimum load. The PTAS combines an optimal schedule of the longest k jobs with the longest processing time (LPT) rule, applied to the remaining jobs.

Formally, for any $k \in [0, n]$, the algorithm A_k is defined as follows:

1. Schedule optimally, with no intended idles, the first k jobs.
2. Add the remaining jobs greedily using the LPT rule.

Theorem 9.1

Let $A_k(I)$ denote the makespan achieved by A_k on an instance I, and let $OPT(I)$ denote the minimum makespan of I, then

$$A_k(I) \leq OPT(I)\left(1 + \frac{1 - \frac{1}{m}}{1 + \lfloor k/m \rfloor}\right)$$

Proof

Let T denote the makespan of an optimal schedule of the first k jobs. Clearly, T is a lower bound for $OPT(I)$, thus, if the makespan is not increased in the second step, i.e., $A_k(I) = T$, then A_k is optimal for I. Otherwise, the makespan of the schedule is greater than T. Let j be the job to determine the makespan (the one which completes last). By the definition of LPT, this implies that all the machines were busy when job j started its execution (otherwise, job j could start earlier). Since the optimal schedule from step 1 has no intended idles, all the machines are busy during the time interval $(0; A_k(I) - p_j)$.

Let $P = \sum_{i=1}^{n} p_i$ be the total processing time of the n jobs. By the above, $P \geq m(A_k(I) - p_j) + p_j$. Also, since the jobs are sorted in nonincreasing order of processing times, we have that $p_j \leq p_{k+1}$, and therefore, $P \geq mA_k(I) - (m-1)p_{k+1}$. A lower bound for the optimal solution is the makespan of a schedule in which the load on the m machines is perfectly balanced; thus, $OPT(I) \geq P/m$, which implies that $A_k(I) \leq OPT(I) + (1 - \frac{1}{m})p_{k+1}$.

To bound $A_k(I)$ in terms of $OPT(I)$, we need to bound p_{k+1} in terms of $OPT(I)$. To obtain such a bound, consider the $k+1$ longest jobs. In an optimal schedule, some machine is assigned at least $\lceil (k+1)/m \rceil \geq 1 + \lfloor k/m \rfloor$ of these jobs. Since each of these jobs has processing time at least p_{k+1}, we conclude that $OPT(I) \geq (1 + \lfloor k/m \rfloor)p_{k+1}$, which implies that $p_{k+1} \leq OPT(I)/(1 + \lfloor k/m \rfloor)$. It follows that

$$A_k(I) \leq OPT(I)\left(1 + \frac{1 - \frac{1}{m}}{1 + \lfloor k/m \rfloor}\right) \qquad \square$$

To observe that the above family of algorithms is a PTAS, we relate the value of k to $(1+\varepsilon)$, the required approximation ratio. Given $\varepsilon > 0$, let $k = \lceil \frac{1-\varepsilon}{\varepsilon} m \rceil$. It is easy to verify that the corresponding algorithm A_k achieves approximation ratio at most $(1+\varepsilon)$. Thus, we conclude that for a fixed m, there is a PTAS for the MM problem.

Note that for any fixed k, an optimal schedule of the first k jobs can be found in $O(m^k)$ steps. Applying the LPT rule takes additional $O(n \log n)$ steps. For A_ε, we get that the running time of the scheme is $O(m^{m/\varepsilon})$, i.e., exponential in m (that is assumed to be constant) and $1/\varepsilon$. This demonstrates the basic property of approximation schemes: a clear trade-off between running time and the quality of approximation.

9.2.1.2 Extend All Possible Solutions for Small Subsets

The second technique, of considering all possible subsets, is illustrated in an early PTAS of Sahni for the *knapsack problem* [11]. An instance of the knapsack problem consists of n items, each having a specified size and a profit, and a single knapsack having size B. Denote by $s_i \geq 0$, $p_i \geq 0$ the size and profit associated with item i. The goal is to find a subset of the items such that the total size of the subset does not exceed the knapsack capacity, and the total profit associated with the items is maximized.

The PTAS in Ref. [11] is based on considering all $O(kn^k)$ possible subsets of size at most k, where k is some fixed constant. Each of these subsets is extended to a larger feasible subset by adding more items to the knapsack, using some greedy rule. The best extension among these $O(kn^k)$ candidates is selected to be the output of the scheme. Formally, for any $k \in [0, n]$, the algorithm A_k is defined as follows:

1. (Preprocessing) Sort the items in nonincreasing order of their profit densities, p_i/s_i.
2. For each feasible subset of at most k items.
   (a) Pack the subset in the knapsack.
   (b) Add to the knapsack items in the sorted list one by one, while there is enough available capacity.
3. Select among the packings generated in Step 2, one which maximizes the profit.

Theorem 9.2

Let $P(A_k)$ denote the profit achieved by A_k, and let $P(OPT)$ denote the optimal profit, then

$$P(OPT) \leq P(A_k)\left(1 + \frac{1}{k}\right)$$

Proof

Let OPT be any optimal solution. If $|OPT| \leq k$ we are done, since the subset OPT will be considered in some iteration of Step 2. Otherwise, let $H = \{a_1, a_2, \ldots, a_k\}$ be the set of k most profitable items in OPT. There exists an iteration of A_k in which H is considered. We show that the profit gained by A_k in this iteration yields the statement of the theorem. Consider the list $L_1 = OPT \backslash H = \{a_{k+1}, \ldots, a_x\}$ of the remaining items of OPT, in the order they appear in the sorted list. Recall that, at some point, A_k will try H as the initial set of k packed items. The algorithm will then add greedily items, as long as the capacity constraint allows. If all the items are packed, A_k is clearly optimal; otherwise, at some point there is not enough space for the next item. Let m be the index of the first item in L_1 which is not packed in the knapsack by A_k, i.e., the items $a_{k+1}, \ldots, a_{m-1}$ are packed. The item a_m is not packed because B_e, the remaining empty space at this point, is smaller than s_m. The greedy algorithm packed into the knapsack only items with profit density at least p_m/s_m. At the time that a_m is dropped, the knapsack contains the items from H, the items $a_{k+1}, \ldots, a_{m-1}$ and some items which are not in OPT.

Let G denote the items packed in the knapsack so far by the greedy stage of A_k. All of these items have profit density at least p_m/s_m. In particular, the items in $G \backslash OPT$ that have total size $\Delta = B - (B_e + \sum_{i=1}^{m-1} s_i)$ all have profit density at least p_m/s_m. Thus, the total profit of the items in G is $P(G) \geq \sum_{i=k+1}^{m-1} p_i + \Delta \frac{p_m}{s_m}$. We conclude that the total profit of the items in OPT is

$$\begin{aligned}
P(OPT) &= \sum_{i=1}^{k} p_i + \sum_{i=k+1}^{m-1} p_i + \sum_{i=m}^{|OPT|} p_i \\
&\leq P(H) + \left(P(G) - \Delta \frac{p_m}{s_m}\right) + \left(B - \sum_{i=1}^{m-1} s_i\right)\frac{p_m}{s_m} \\
&= P(H) + P(G) + B_e \frac{p_m}{s_m} < P(H \cup G) + p_m
\end{aligned}$$

Since A_k packs at least $H \cup G$, we get that $P(A_k) \geq P(H) + P(G)$, which implies that $P(OPT) - P(A_k) < p_m$. Given that there are at least k items with a profit at least as large as a_m (those selected to H), we conclude that $p_m \leq P(OPT)/(k+1)$. This gives the approximation ratio. □

Assuming a single preprocessing step, in which the items are sorted by their profit densities, each subset is extended to a maximal packing in time $O(n)$. Since there are $O(kn^k)$ possible subsets to consider, the total running time of the scheme is $O(kn^{k+1})$.

To obtain a PTAS for the knapsack problem, let A_ε be the algorithm A_k with $k = \lceil 1/\varepsilon \rceil$. By the above, the approximation ratio is at most $1 + \varepsilon$, and the running time of A_ε is $O(\frac{1}{\varepsilon} n^{1+\frac{1}{\varepsilon}})$.

The technique of choosing the best among a small number of partial packings was applied also to variants of multidimensional packing. A detailed example is given in Section 9.3.2.

9.2.2 Applying Enumeration to a Compacted Instance

In this section we present the technique of applying exhaustive enumeration to a modified instance, in which we have a more compact representation of the input. Approximation schemes that are based on this approach consist of three steps:

1. The instance I is modified to a simpler instance, I'. The parameter ε determines how rough I' is compared with I. The smaller ε, the more refined is I'.
2. The problem is solved optimally on I'.
3. An approximate solution for I is induced from the optimal solution for I'.

The challenge is to modify I in the first step into an instance I' that is simple enough to be solved in polynomial time, yet not too different from the original I, so that we can use an exact solution for I' to derive an approximate solution for I.

The use of this technique usually involves partitioning the input into *significant* and *nonsignificant* elements. The partition depends on the problem at hand. For example, it is natural to distinguish between *long* and *short* jobs in scheduling problems, and between *big* and *small*, or *high-profit* and *low-profit* elements, in packing problems. For a given instance, the distinction between the two types of elements usually depends on the input parameters (including ε), and on the optimal solution value.

In some cases, the transformation from I to I' involves only grouping the nonsignificant elements. Each group of such elements thus forms a single significant element in I'. As a result, the instance I' consists of a small number of significant elements. More details and an example for this type of transformation are given in Section 9.2.2.1.

In other cases, all the elements, or only the more significant ones, are transformed into a set of elements with a small number of distinct values. This approach is described and demonstrated in Section 9.2.2.2.

9.2.2.1 Grouping Subsets of Elements

We illustrate the technique with the PTAS of Sahni [12] for the MM problem on two identical machines. The input consists of n jobs with processing times $p_1, \ldots, p_n$. The goal is to schedule the jobs on two identical parallel machines in a way that minimizes the latest completion time. In other words, we seek a schedule which balances the load on the two machines as much as possible.

Let $P = \sum_{j=1}^{n} p_j$ denote the total processing time of all jobs, and let p_{max} denote the longest processing time of a job. Let $C = \max(P/2,\ p_{max})$. Note that C is a lower bound on the MM (i.e., $OPT \geq C$), since $P/2$ is the schedule length if the load is perfectly balanced between the two machines, and since some machine must process the longest job.

The first step of the scheme is to modify the instance I into a simplified instance I'. This modification depends on the value of C and on the parameter ε. Given I, ε, partition the jobs into small jobs—of length at most εC, and big jobs—of length greater than εC. Let P_S denote the total length of small jobs. The modified instance I' consists of the big jobs in I together with $\lfloor P_S/(\varepsilon C) \rfloor$ jobs of length εC.

Next, we need to solve optimally the MM problem for the instance I'. Note that all jobs in I' have length at least εC and their total size is at most P, the total processing time of the jobs in the original instance, since the small jobs in I are replaced in I' by jobs of length εC with total length at most P_S. Therefore, the number of jobs in I' is at most *the constant* $P/\varepsilon C \leq 2/\varepsilon$. An optimal schedule of a constant number of jobs can be found by exhaustive search over all $O(2^{2/\varepsilon})$ possible schedules. This constant number is independent of n, but grows exponentially with ε, as we expect from our PTAS.

Finally, we need to transform the optimal schedule of I' into a feasible schedule of I. Note that, for the makespan objective, we are only concerned about the partition of the jobs between the machines, while the order in which the jobs are scheduled on each machine can be arbitrary. Denote by $OPT(I')$ the length of the optimal schedule for I'. To obtain a schedule of I, each of the big jobs is scheduled on the same machine as in the optimal schedule for I'. The small jobs are scheduled greedily in an arbitrary order on the first machine until, for the first time, the total load on the first machine is at least $OPT(I')$. The remaining small jobs are scheduled on the second machine. Clearly, the overflow on the first machine is at most εC (maximal length of a small job). Also, since the total number of (εC)-jobs was defined to be $\lfloor P_S/(\varepsilon C)\rfloor$, the overflow on the second machine is also bounded by εC. Therefore, the resulting makespan in the schedule of I is at most $OPT(I') + \varepsilon C$.

To complete the analysis we need to relate $OPT(I')$ to $OPT(I)$.

Claim 9.1

$OPT(I') \leq (1+\varepsilon)OPT(I)$

Proof

Given a schedule of I, in particular an optimal one, a schedule for I' can be derived by replacing—on each machine separately—the small jobs with jobs of size εC, with at least the same total size. Recall that the number of (εC)-jobs in I' is $\lfloor P_S/(\varepsilon C)\rfloor$. Regardless of the partition of the small jobs in I between the two machines, the result of this replacement is a feasible schedule of I' whose makespan is at most $OPT(I) + \varepsilon C$. Since $OPT(I) \geq C$, the statement of the claim holds. □

Back to our scheme, we showed that the optimal schedule of I' is transformed into a feasible schedule of I whose makespan is at most $OPT(I') + \varepsilon C$. By Claim 9.1, this value is at most $(1+\varepsilon)OPT(I) + \varepsilon C \leq (1+2\varepsilon)OPT(I)$. By selecting $\varepsilon' = \varepsilon/2$, and running the scheme with ε', we get the desired ratio of $(1+\varepsilon)$.

The above scheme can be extended to any constant number of machines. For *arbitrary* number of machines, a more complex PTAS exists: the scheme of Ref. [1], which requires reducing the number of distinct values in the input, is given in the next section.

9.2.2.2 Reducing the Number of Distinct Values in the Input

Any optimization problem can be solved optimally in polynomial, or even constant, time if the input size is some constant. For many optimization problems, an efficient algorithm exists if the input size is arbitrary, but the number of *distinct values* in the input is some constant. Alternatively, the problem can be solved by a pseudopolynomial-time algorithm (e.g., by dynamic programming), whose running time depends on the instance parameters, and is therefore polynomial only if the parameter values are polynomial in the problem size.

The idea behind the technique that we describe below is to transform the elements (or sometimes, only the significant elements) in the instance I into an instance I' in which the number of distinct values is fixed, or to scale the values according to the input size. The problem is then solved on I', and the solution for I' is transformed into a solution for the original instance. The nonsignificant elements, which are sometimes omitted from I', are added later to the solution, using some heuristic. The parameter ε determines the (constant) number of distinct values contained in I': the smaller the ε, the larger the number of distinct values. The following are the two main approaches for determining the values in I'.

1. *Rounding*. The values in I' form an arithmetic series in which the difference between elements is a function of ε. For example, multiples of $\varepsilon^2 T$, for some value T. In this approach, the gap between any two values bounds the difference between the original value of an element in I and the value of the corresponding element in I'. Note that the number of elements whose values are rounded to a single value in I' can be arbitrary.

2. *Shifting.* The values in I' are a subset of the values in I, selected such that the distribution on the number of values in I that are shifted to a single value in I' is uniform. However, in contrast to the rounding approach, there is no bound on the difference between the value of an element in I and its value in I'. For example, partition the elements into $\lceil 1/\varepsilon^2 \rceil$ groups, each having at most $\lfloor n/\varepsilon^2 \rfloor$ elements, and fix the values in each group to be (say) the minimal value of any element in the group.

In both approaches, the approximation ratio is guaranteed to be $(1+\varepsilon)$ if I' is close enough to I. Formally, an optimal solution for I' induces a solution for I whose value is greater/smaller by a factor of at most $(1+\varepsilon)$. Another factor of $(1+\varepsilon)$ may be added to the approximation ratio due to the nonsignificant items—in case they are handled separately.

We demonstrate this technique with the classic PTAS of Hochbaum and Shmoys [1] for the MM problem on parallel machines. The input for the problem is a set of n jobs having processing times $p_1, \ldots, p_n$, and m identical machines; the goal is to schedule the jobs on the machines in a way that minimizes the latest completion time of any job. The number of machines, m, can be arbitrarily large (otherwise, a simpler PTAS exists; see Section 9.2.2.1).

First, note that the MM problem is closely related to the *bin packing* (BP) problem. The input for BP is a collection of items whose sizes are in (0, 1). The goal is to pack all items using a minimal number of bins. Formally, let $I = \{p_1, \ldots, p_n\}$ be the sizes in a set of n items, where $0 \le p_j \le 1$. The goal is to find a collection of subsets $U = \{B_1, B_2, \ldots, B_k\}$ which forms a disjoint partition of I, such that for all i, $1 \le i \le k$, $\sum_{j \in B_i} p_j \le 1$, and the number of bins, k, is minimized.

The exact solutions of MM and BP relate in the following way. It is possible to schedule all the jobs in an MM instance on m machines with makespan C_{max} if and only if it is possible to pack all the items in a BP instance, where the size of item j is p_j/C_{max}, in m bins. The relation between the optimal solutions does not remain valid for approximations. In particular, BP admits an asymptotic FPTAS (see Chapter 11), while MM does not. However, this relation can be used to develop a PTAS for MM.

Let $OPT_{BP}(I)$ be the number of bins in an optimal solution of BP, and let $OPT_{MM}(I) = C_{max}$ be an optimal solution for MM. Denote by $\frac{I}{d}$ the BP input in which all the values are divided by d. We already argued that

$$OPT_{BP}\left(\frac{I}{d}\right) \le m \;\Leftrightarrow\; OPT_{MM}(I, m) \le d$$

We define a *dual* approximation scheme for BP. For an input I, we seek a solution with at most OPT_{BP} bins, where each bin is filled to capacity at most $1+\varepsilon$. In other words, we relax the bin capacity constraint by a factor of $1+\varepsilon$. Let $dual_\varepsilon(I)$ be such an algorithm, and let $DUAL_\varepsilon(I)$ be the number of bins in the corresponding packing.

Theorem 9.3

If there exists a dual approximation algorithm for BP, then there is a PTAS for the MM problem.

Proof

The PTAS performs a binary search to find OPT_{MM}. To bound the range in which the optimal makespan is searched, two lower bounds and one upper bound for this value are used. The lower bounds are the length of the longest job and the load on each machine when the total load is perfectly balanced. That is, let $SIZE(I, m) = max\{\frac{1}{m}\sum p_i, p_{max}\}$, then $OPT_{MM} \ge SIZE(I, m)$. The upper bound uses the fact that the simple *list scheduling* algorithm attains a 2-ratio to $SIZE(I, m)$ [10], therefore $OPT_{MM} \le 2SIZE(I, m)$.

Now it is possible to perform a binary search to find OPT_{MM}. Instead of checking whether $OPT_{MM} < d$, the algorithm checks whether $DUAL_\varepsilon(\frac{I}{d}) < m$.

```
$upper = 2SIZE(I, m)$
$lower = SIZE(I, m)$
repeat until $lower = upper$
    $d = (lower + upper)/2$
    call $dual_\varepsilon(\frac{I}{d})$
    if $DUAL_\varepsilon(\frac{I}{d}) > m$
        $lower \leftarrow d$
    else
        $upper \leftarrow d$
$d^\star \leftarrow upper$
return $dual_\varepsilon(\frac{I}{d^\star})$
```

Initially, $OPT_{MM}(I, m) \leq upper \Rightarrow OPT_{BP}(\frac{I}{upper}) \leq m$. Since $dual_\varepsilon$ is a relaxation of BP, $DUAL_\varepsilon(\frac{I}{upper}) \leq OPT_{BP}(\frac{I}{upper})$. This implies that $DUAL_\varepsilon(\frac{I}{upper}) \leq m$. By the update rule, the above remains true during the execution of the loop. However,

$$DUAL_\varepsilon\left(\frac{I}{upper}\right) \leq m \Rightarrow OPT_{MM}(I, m) \leq (1+\varepsilon)upper$$

and thus $(1+\varepsilon)upper$ remains an upper bound on $OPT_{MM}(I, m)$ during the search. Similarly, before the loop $OPT_{MM}(I, m) \geq lower$, which remains true since $DUAL_\varepsilon(\frac{I}{lower}) \geq m$, is an invariant of the loop, and

$$OPT_{BP}\left(\frac{I}{lower}\right) \geq DUAL_\varepsilon\left(\frac{I}{lower}\right) \geq m \Rightarrow OPT_{MM}(I, m) \geq lower$$

Thus, the solution value is bounded above by

$$(1+\varepsilon)\cdot d^\star = (1+\varepsilon)\cdot upper = (1+\varepsilon)\cdot lower \leq (1+\varepsilon)OPT_{MM}(I, m)$$

In practice, assume that we stop the binary search after k iterations. At this time, it is guaranteed that $upper - lower \leq 2^{-k}SIZE(I, m) \leq 2^{-k}OPT_{MM}(I, m)$, and the value of the solution is bounded above by $(1+\varepsilon)\cdot d^\star = (1+\varepsilon)\cdot upper \leq (1+\varepsilon)\cdot(lower + 2^{-k}OPT_{MM}(I, m)) \leq (1+\varepsilon)(1+2^{-k})OPT_{MM}(I, m)$.

By choosing $k = O(\log\frac{1}{\epsilon})$, and taking in the scheme $\varepsilon' = \varepsilon/3$, we obtain a $(1+\varepsilon)$-approximation. □

We now describe the $dual_\varepsilon$ approximation scheme for BP. This scheme uses the rounding and grouping technique.

Theorem 9.4

There exists an $O\left(n^{\lceil\frac{1}{\varepsilon^2}\rceil}\right)$-time dual approximation scheme for BP.

Proof

Recall that, for a given $\varepsilon > 0$, the dual approximation scheme needs to find a packing of all items using at most OPT_{BP} bins, such that the total size of the items packed in each bin is at most $1+\varepsilon$. The basic idea is to omit first the "small" items and then round the sizes of the "big" items; this yields an instance in which the number of distinct item sizes is fixed. We can now solve the problem exactly using dynamic programming, and the solution induces a solution for the original instance, where each bin is filled up to capacity $1+\varepsilon$.

The first observation is that small items, whose sizes are less than ε, can be initially omitted. The problem will be solved for big items only and the small items will be added later on greedily, in the following manner: if there is a bin filled with items of total size less than 1, small items are added to it; otherwise, a new bin is opened. If no new bin is opened, then, clearly, no more than the optimum number of bins is used (as the dual PTAS uses the minimal number of bins for the big items). If new bins were added, then all original bins are filled to capacity at least 1, and all the new bins (except maybe the last one) are also filled to capacity at least 1. This is optimal since $OPT(I) \geq \lceil\sum p_i\rceil \geq DUAL_\varepsilon(I)$. We conclude that, without loss of generality, all items are of size $\varepsilon \le p_i \le 1$. Divide the range $[\varepsilon, 1]$ into intervals of size ε^2. This gives $S = \lceil \frac{1}{\varepsilon^2} \rceil$ intervals. Denote by l_i the endpoints of the intervals and let b_i be the number of elements whose sizes are in the interval $(l_i, l_{i+1}]$.

We now examine a packed bin. Since the minimal item size is ε, the bin can contain at most $\lfloor \frac{1}{\epsilon} \rfloor$ items. Denote by X_i the number of items in the bin whose sizes are in the interval $(l_i, l_{i+1}]$. X_i is in the range $[0, \lfloor \frac{1}{\epsilon} \rfloor)$. Let the vector $(X_1, \dots, X_S)$ denote the *configuration* of the bin. The number of feasible configurations is bounded above by $\lfloor \frac{1}{\epsilon} \rfloor^S$. A configuration is *feasible* if and only if $\sum_{i=1}^{S} X_i l_i \le 1$.

For any bin B whose packing forms a feasible configuration, the total size of the items in the bin is bounded by

$$\sum_{j \in B} p_j \le \sum_{j \in B} X_j l_{j+1} \le \sum_{j \in B} X_j (l_j + \varepsilon^2) \le 1 + \varepsilon^2 \sum_{j \in B} X_j \le 1 + \varepsilon^2 \cdot \frac{1}{\epsilon} \le 1 + \varepsilon$$

Therefore, it is sufficient to solve the instance with all item sizes rounded down to sizes in $\{l_1, \dots, l_S\}$.

Finally, we describe a dynamic programming algorithm which solves the BP problem exactly when the number of distinct item sizes is fixed. Let $BINS(b_1, b_2, \dots, b_S)$ be the minimal number of bins required to pack b_1 items of size l_1, b_2 items of size l_2, …, and b_S items of size l_S. Let $\mathcal{C}$ denote the set of all feasible configurations. Observe that, by a standard dynamic programming recursion,

$$BINS(b_1, b_2, \dots, b_S) = 1 + min_{\mathcal{C}} BINS(b_1 - X_1, b_2 - X_2, \dots, b_S - X_S)$$

We minimize over all possible vectors $(X_1, X_2, \dots, X_s)$ that correspond to a feasible packing of the "first" bin (counted by the constant 1), and the best way to pack the remaining items (this is the recursive call). Thus, the dynamic programming procedure builds a table of size n^S, where the calculation of each entry requires $O(\lfloor \frac{1}{\epsilon} \rfloor^S)$.

This yields a running time of

$$O\left(n^S \cdot \lfloor \frac{1}{\epsilon} \rfloor^S\right) = O\left(\left(\frac{n}{\varepsilon}\right)^{\lceil \frac{1}{\varepsilon^2} \rceil}\right) = O\left(n^{\lceil \frac{1}{\varepsilon^2} \rceil}\right)$$

□

The technique of applying enumeration to a compacted instance through grouping/rounding has been extensively used in PTASs for scheduling problems (see, e.g., Refs. [13–15]). A common approach for compacting the instance is to reduce the input parameters to *poly bounded*, i.e., parameters whose values can be bounded as function of the input size. This approach is used, e.g., in the PTAS of Chekuri and Khanna for preemptive weighted flow time [4] (See the survey paper [9]).

9.2.3 More on Grouping and Shifting

In the following we outline two extensions of the techniques described in this section.

Randomized Grouping

In some cases, we need to define a partition of the input elements to groups $(I_1, \dots, I_k)$, using for each element x a parameter of the problem, $q(x)$, such that the elements in two groups I_j and I_{j+1} differ in their $q(x)$ value by roughly a factor of α, for some $\alpha > 1$. When such partition is infeasible, we can use randomization to achieve an *expected* separation between groups. For a parameter $\alpha > 1$, the following randomized geometric grouping technique yields an expected separation that is logarithmic in α. This technique extends the deterministic geometric rounding technique described in Section 9.2.2.2. Initially, pick a number $r \in [1, \alpha]$ at random, by a probability distribution having the density function $f(y) = 1/y \ln \alpha$. An element x with the value $q(x)$ belongs to the group I_j if $q(x) \in [r\alpha^j, r\alpha^{j+1}]$. Thus, the index of the group to which x belongs, denoted by $g(x)$, is a random variable which can take two possible values: $\lfloor \log_\alpha q(x) \rfloor$ or $\lfloor \log_\alpha q(x) \rfloor + 1$. It can be shown that for a fixed α, the number of distinct partitions induced by the random choices of r is at most the number of elements in the input. This enables to easily derandomize algorithms that use randomized geometric grouping. The technique was applied, e.g., by Chekuri and Khanna [4] in a PTAS for preemptive weighted flow time.

Oblivious Shifting

While applying the standard shifting technique (as described in Section 9.2.2.2) requires knowing the initial input parameters, it is possible to apply shifting also when not all values are known a-priori. In *oblivious shifting*, the input size is initially known, and the scheme starts by defining the number of values in the resulting instance, but the actual shifted values are revealed at a later stage, by optimizing on these values, considering the constraints of the problem. The technique can be used for defining a "good" compacted instance from a partial solution for the problem, which can then enable to obtain a complete solution for the problem efficiently.

For example, a variant of the BP problem, in which items may be *fragmented*, is solved in Ref. [16] in two steps. Given the input, we need to determine the set of items that will be fragmented, as well as the fragment sizes in a feasible approximate solution. Since the possible number of fragment sizes is large, a compact vector of fragments is generated, which contains a bounded number of *unknown* shifted fragment sizes. The actual sizes of the shifted fragments are determined by solving a linear program (LP) which attempts to find a feasible packing of these fragments. A detailed description is given in Ref. [16].

9.3 Rounding Linear Programs

In this section we discuss approximations obtained using linear programming relaxation of the integer program formulation of a given optimization problem. We refer the reader to Chapters 6 and 7 of this handbook for further background on linear programming and rounding linear programs. Most generally, the technique is based on solving a linear programming relaxation of the problem, for which an exact or approximate solution can be obtained efficiently. This solution is then rounded, thus yielding an approximate integral solution. The (fractional) solution obtained for the LP needs to have some nice properties that would allow rounding to be not too harmful, in terms of ε, the accuracy parameter of the scheme. One such property of an LP, which is commonly used, is the existence of a small *basic solution*. We illustrate below the usage of this property, with examples from *vector scheduling (VS)* and covering integer programs. An LP has a *small* basic solution, if there exists an optimal solution in which the number of nonzero variables is small as a function of the input size and ε. For such a solution, the error incurred by rounding can be bounded, such that the resulting integral solution is within factor of $1 + \varepsilon$ from the optimal. A natural example is the class of LPs in which either the number of variables or the number of constraints is some fixed constant. For such programs, there exists a basic solution in which the number of nonzero variables is fixed; however, depending on the problem, and in particular, on the value of an optimal solution for the LP, a basic solution can be "small," even if the number of nonzero variables is relatively large, for example, $\Omega(\varepsilon n)$, where n is the number of variables.

LP rounding can be combined with the techniques described in Section 9.2. In Section 9.3.1 we show the usage of LP rounding for a given subset of input elements satisfying certain properties. In Section 9.3.2 we show how LP rounding can be combined with the selection of *all* possible (small) subsets.

9.3.1 Solving LP for a Subset of Elements

As mentioned earlier, in many problems, an approximation scheme can be obtained by partitioning a set of input elements to subsets, and solving the problem for each subset separately. For some subsets, a good solution can be obtained by rounding an LP relaxation of the problem.

In certain assignment problems, we can find an almost integral basic solution for an LP, for part of the input, since the relation between the number of variables and nontrivial constraints in the linear programming relaxation, combined with the assignment requirement of the problem, imply that only few variables can get fractional values. This essential property is used, e.g., in the PTAS of Chekuri and Khanna for the *VS* problem [17]. The VS problem is to schedule d-dimensional jobs on m identical machines, such that the maximum load over all dimensions and over all machines is minimized. Formally, an instance I of VS consists of n jobs, $J_1, \ldots, J_n$, where J_j is associated with a rational d-dimensional vector $(p_j^1, \ldots, p_j^d)$, and m machines. We need to assign the jobs to the machines, i.e., schedule a subset of the jobs, A_i, on machine i, $1 \le i \le m$, such that $\max_{1 \le i \le m} \max_{1 \le h \le d} \sum_{J_j \in A_i} p_j^h$ is minimized.

Note that in the special case where $d = 1$, we get the minimum makespan problem (see Section 9.2.2.2). The PTAS in Ref. [17] for the VS problem, where d is fixed, applies a nontrivial generalization of the PTAS of Hochbaum and Shmoys for the case $d = 1$ [1]. The scheme is based on a primal-dual approach, in which the primal problem is VS and the dual problem is vector packing. Thus, the machines are viewed as d-dimensional bins, and the schedule length as bin capacity (or height). W.l.o.g., we may assume that the optimal schedule has the value 1. Given an $\varepsilon > 0$ and a correct guess of the optimal value, we describe below an algorithm A_ε that returns a schedule of height at most $1+\varepsilon$. Arriving at the correct guess involves a binary search for the optimal value (which can be done in polynomial time; see below).

Let $\delta = \varepsilon/d$ be a parameter. The scheme starts with a preprocessing step, which enables to bound the ratio of the largest coordinate to the smallest nonzero coordinate in any input vector. Specifically, let $\| J_j \|_\infty = \max_{1 \le h \le d} p_j^h$ be the ℓ_∞ norm of J_j, $1 \le j \le n$, then, for any J_j, and any $1 \le h \le d$, if $p_j^h \le \delta \cdot \| J_j \|_\infty$, we set $p_j^h = 0$. As shown in Ref. [17], any valid schedule for the resulting modified instance, I', yields a valid solution for the original instance, I, whose height is at most $(1+\varepsilon)$ times that of I'.

We consider from now on only transformed instances. The scheme proceeds by partitioning the jobs to the sets L (*large*) and S (*small*). The set L consists of all vectors whose ℓ_∞ norm is greater than δ, and S contains the remaining vectors. The algorithm A_ε packs first the large jobs and then the small jobs. Note that while in the case of $d = 1$ these packings are done independently, for $d \ge 2$, we need to consider the interaction between these two sets. Similar to the scheme of Hochbaum and Shmoys [1], a valid schedule is found for the jobs by guessing a configuration. In particular, let the d-tuple $(a_1, \ldots, a_d)$ $0 \le a_h \le \lceil 1/\varepsilon \rceil$, $1 \le h \le d$, denote a *capacity configuration*, that is, the way some bin is filled. Since $d \ge 2$ is a constant, the possible number of capacity configurations, given by $W = (1 + \lceil 1/\varepsilon \rceil)^d$, is also a constant. Then, by numbering the capacity configurations, we describe by a W-tuple $M = (m_1, \ldots, m_W)$ the number of bins having capacity configuration w, where $1 \le w \le W$. The possible number of *bin configurations* is then $O(m^W)$. This allows to guess a bin configuration which yields the desired $(1 + \varepsilon)$-approximate solution in polynomial time.

We say that a packing of vectors in a bin *respects* a capacity configuration $(a_1, \ldots, a_d)$ if the height of the packing is smaller than εa_h for any $1 \le h \le d$. Given a capacity configuration $(a_1, \ldots, a_d)$, we define the *empty capacity configuration* to be the d-tuple $(\bar{a}_1, \ldots, \bar{a}_d)$, where $\bar{a}_h = \lceil 1/\varepsilon \rceil + 1 - a_h$, for $1 \le h \le d$. For a given bin configuration, M, we denote by $\bar{M}$ the bin configuration obtained by taking for each of the bins in M the corresponding empty capacity configuration.

The scheme performs the following two steps for each possible bin configuration, M: (i) decides whether vectors in L can be packed respecting M, and (ii) decides whether vectors in S can be packed respecting $\bar{M}$. Given that we have guessed the correct bin configuration M, both steps will succeed, and we get a packing of height at most $1 + \varepsilon$.

We now describe how the scheme packs the large and the small vectors. The vectors in L are packed using rounding and dynamic programming. In particular, since by definition, any entry in a vector in L has the value δ^2 or greater, we use geometric rounding, that is, for each vector J_j, and any entry p_j^h, $1 \le h \le d$, p_j^h is rounded down to the nearest value of the form $\delta^2(1 + \varepsilon)^t$, for $0 \le t \le \lceil \frac{2}{\varepsilon} \log 1/\delta \rceil$. Denote the resulting set of vectors L', and the modified instance I'. The vectors in L' can be partitioned into

$$q = \left(1 + \lceil \frac{2}{\varepsilon} \log 1/\delta \rceil\right)^d \tag{9.1}$$

classes. The proofs of the next lemmas are given in Ref. [17].

Lemma 9.1

Given a solution for I', replacing each vector in L' by the corresponding vector in L results in a valid solution for I whose height is at most $1 + \varepsilon$ times that of I'.

Lemma 9.2

Given a correct guess of a bin configuration M, there exists an algorithm which finds a packing of the vectors in L' that respects M, and whose running time is $O((d/\delta)^q mn^q)$, where q is given in Eq. (9.1).

The small vectors are packed using a linear programming relaxation and careful rounding. Renumber the vectors in S by $1, \ldots, |S|$. Let $x_{ji} \in \{0, 1\}$ be an indicator variable for the assignment of the vector J_j to machine i, $1 \leq j \leq n$, $1 \leq i \leq m$. In the LP relaxation $x_{ji} \geq 0$. We solve the following LP.

$$(LP) \qquad \sum_{J_j \in S} p_j^h x_{ji} \leq b_i^h, \qquad 1 \leq i \leq m, \quad 1 \leq h \leq d \tag{9.2}$$

$$\textstyle\sum_{i=1}^m x_{ji} = 1, \qquad 1 \leq j \leq |S| \tag{9.3}$$

$$x_{ji} \geq 0, \qquad 1 \leq j \leq n, \quad 1 \leq i \leq m \tag{9.4}$$

The constraints (9.2) guarantee that the packing does not exceed a given height bound in any dimension (i.e., the available height after packing the large vectors). The constraints (9.3) reflect the requirement that each vector is assigned to one machine. A key property of the LP, which enables to obtain an integral solution that is close to the fractional, is given in the next result.

Lemma 9.3

In any basic feasible solution for LP, at most $d \cdot m$ vectors are assigned (fractionally) to more than one machine.

Proof

Recall that the number of nonzero variables, in any *basic* solution for an LP, is bounded by the number of tight constraints in some optimal solution (since nontight constraints can be omitted). Since the number of nontrivial constraints (i.e., constraints other than $x_{ji} \geq 0$) is $(|S| + d \cdot m)$, it follows that the number of strictly positive variables in any basic solution is at most $(|S| + d \cdot m)$. Since each vector is assigned to at least one machine, the number of vectors which are fractionally assigned to more than one machine is at most $d \cdot m$. □

The above type of argument was first made and exploited by Potts [18] in the context of parallel machine scheduling. It was later applied to other problems, such as job shop scheduling (see, e.g., Ref. [19]).

Thus, we solve the above program and obtain a basic solution. Denote by S' the set of vectors which are assigned fractionally to two machines or more. Since $|S'| \leq d \cdot m$, we can partition the set S' to subsets of size at most d each, and schedule the ith set to the ith machine. Since $\| J_j \|_\infty \leq \delta = \varepsilon/d$, for all $J_j \in S'$, the total height of the machines is violated at most by ε in any dimension. We can therefore summarize in the following theorem.

Theorem 9.5

For any $\varepsilon > 0$, there is a $(1 + \varepsilon)$-approximation algorithm for VS whose running time is $(nd/\varepsilon)^{O(f)}$, where $f = O((\frac{\ln(d/\varepsilon)}{\varepsilon})^d)$.

Proof

By the above discussion, given the correct guess of the optimal value, the scheme yields a schedule of value (height) at most $1 + O(\varepsilon)$ the optimal. We need to find a packing of the vectors in L and S, for each bin configuration M. The running time for a single configuration is dominated by the packing of L, and since the number of configurations is $m^W = O(n^{O(1/\varepsilon^d)})$, we get the running time from Lemma 9.2. The value of an optimal schedule can be guessed, within factor $1 + \varepsilon$, by obtaining first a $(d + 1)$-approximate solution. This can be done by applying an approximation algorithm for resource constrained scheduling due to Ref. [20]. □

9.3.2 LP Rounding Combined with Enumeration

As described in Section 9.2.1, a common technique for obtaining a PTAS is to extend all possible solutions for small subsets of elements. This technique can be combined with LP rounding as follows. Repeatedly select a small subset of input elements, $S_g \subseteq I$, to be the basis for an approximate solution; solve an LP for the remaining elements, $I \setminus S_g$. Select the subset S_g which gives the best solution. We exemplify the usage of the technique to obtain a PTAS for *covering integer programs with multiplicity constraints (CIP)*. In this core problem, we must fill up an R-dimensional bin by selecting (with bounded number of repetitions) from a set of n R-dimensional items, such that the overall cost is minimized. Formally, let $A = \{a_{ji}\}$ denote the sizes of the items in the R dimensions, $1 \le j \le R$, $1 \le i \le n$; the cost of item i is $c_i \ge 0$. Let x_i denote the number of copies selected from item i, $1 \le i \le n$. We seek an n-vector $\mathbf{x}$ of nonnegative integers, which minimizes $c^T\mathbf{x}$, subject to the R constraints given by $A\mathbf{x} \ge b$, where $b_j \ge 0$ is the size of the bin in dimension j. In addition, we have multiplicity constraints for the vector $\mathbf{x}$, given by $\mathbf{x} \le d$, where $d \in \{1, 2, \ldots\}^n$.

Covering integer programs form a large subclass of integer programs encompassing such NP-hard problems as minimum knapsack and set cover. This implies the hardness of CIP in fixed dimension (i.e., where R is a fixed constant). For general instances, the hardness of approximation results for set cover carry over to CIP. Comprehensive surveys of known results for CIP and CIP_∞, where the multiplicity constraints are omitted, are given in Refs. [21,22] (see also in Ref. [23]).

We describe below a PTAS for CIP in fixed dimension. The scheme presented in Ref. [21] builds on the classic LP-based scheme due to Frieze and Clarke for the R-dimensional knapsack problem [24]. Consider an instance of CIP in fixed dimension, R. We want to minimize $\sum_{i=1}^{n} c_i x_i$ subject to the constraints $\sum_{i=1}^{n} a_{ij} x_i \ge b_j$ for $j = 1, \ldots, R$, and $x_i \in \{0, 1, \ldots d_i\}$ for $i = 1, \ldots, n$.

Assume that we know the optimal cost, C, for the CIP instance. The scheme of Ref. [21] uses a reduction to the binary *minimum R-dimensional multiple choice knapsack (R-MMCK)* problem. For some $R \ge 1$, an instance of binary R-MMCK consists of a single R-dimensional knapsack, of size b_j in the jth dimension, and m sets of items. Each item has an R-dimensional size and is associated with a cost. The goal is to pack a subset of items, by selecting at most one item from each set, such that the total size of the packed items in dimension j is at least b_j, $1 \le j \le R$, and the overall cost is minimized.

Given the value of C, the parameter ε, and a CIP instance with bounded multiplicity, the scheme constructs an R-MMCK instance in which the knapsack capacities in the R dimensions are b_j, $1 \le j \le R$. Also, there are n sets of items denoted by A^i, $1 \le i \le n$. Let $\hat{K}^i$ be the integer value satisfying $d_i c_i \in [\hat{K}^i \varepsilon C/n, (\hat{K}^i + 1)\varepsilon C/n)$, then the number of items in A^i is $K^i = \min(\hat{K}^i, \lfloor n/\varepsilon \rfloor)$. The set A^i represents all possible values which x_i can take in the solution for CIP. In particular, the kth item in A^i, denoted (i, k), represents the assignment of a value in $[0, d_i]$ to x_i, such that $c(i, k)$, the total cost incurred by item i is in $[k\varepsilon C/n, (k + 1)\varepsilon C/n)$. This total cost is rounded down to the nearest integral multiple of $\varepsilon C/n$; thus, $c(i, k) = k\varepsilon C/n$. The size of the item (i, k) in dimension j, $1 \le j \le R$, is given by $s_j(i, k) = a_{ij}$.

Given an instance of R-MMCK, guess a partial solution, given by a small size set, S; these items have the maximal costs in some optimal solution. The size of S is a fixed constant, namely, $|S| = h = \lfloor \frac{2R(1+\varepsilon)}{\varepsilon} \rfloor$. The set S will be extended to an *approximate* solution, by solving an LP for the remaining items. The value of h is chosen such that the resulting solution is guaranteed to be within $1 + \varepsilon$ from the optimal, as computed below. Let $E(S)$ be the subset of items with costs that are larger than the minimal cost of any item in S, that is, $E(S) = \{(i, k) \notin S \mid c(i, k) > c_{min}(S)\}$, where $c_{min}(S) = \min_{(i,k)\in S} c(i, k)$. Select all the items $(i, k) \in S$, and eliminate from the instance all the items $(i, k) \in E(S)$ and the sets A^i from which an item has been selected. In the next step we find an optimal *basic solution* for the following LP, in which $x_{i,k}$ is an indicator variable for the selection of the item (i, k).

$$
\begin{aligned}
(LP(S)) \quad & \text{minimize} && \sum_{i=1}^{n}\sum_{k=1}^{K^i} x_{i,k} \cdot c(i, k) \\
& \text{subject to} && \sum_{k=1}^{K^i} x_{i,k} \le 1, \quad \text{for } i = 1, \ldots, n, \\
& && \sum_{i=1}^{n}\sum_{k=1}^{K^i} s_j(i, k) x_{i,k} \ge b_j \quad \text{for } j = 1, \ldots, R \\
& && 0 \le x_{i,k} \le 1 \quad \text{for } (i, k) \notin S \cup E(S) \\
& && x_{i,k} = 1 \quad \text{for } (i, k) \in S \\
& && x_{i,k} = 0 \quad \text{for } (i, k) \in E(S)
\end{aligned}
$$

Given an optimal fractional solution for the above program, we get an integral solution as follows. For any i, $1 \leq i \leq n$, let $k_{max} = k_{max}(i)$ be the maximal value of $1 \leq k \leq K^i$ such that $x_{i,k} > 0$, then we set $x_{i,k_{max}} = 1$ and, for any other item in A^i, $x_{i,k} = 0$. Finally, we return to the CIP instance and assign to x_i the maximum value for which the total (rounded down) cost for item i is $c(i, k_{max})$.

The next three lemmas show that the scheme yields a $(1 + \varepsilon)$-approximation to the optimal cost, and that the resulting integral solution is feasible.

Lemma 9.4

If there exists an optimal (integral) solution for CIP with cost C, then the integral solution obtained from the rounding for R-MMCK has the cost $\hat{z} \leq (1 + \varepsilon)C$.

Proof

Let $\mathbf{x}^*$ be an optimal (fractional) solution for the linear program LP(S), and let S^* be the corresponding subset of items, that is, $S^* = \{(i, k)|\ x^*_{i,k} = 1\}$. If $|S^*| < h$ then we are done: in some iteration, the scheme will try S^*; otherwise, let $S^* = \{(i_1, k_1), \ldots, (i_g, k_g)\}$, such that $c(i_1, k_1) \geq \cdots \geq c(i_g, k_g)$, for some $g > h$. Let $S^*_h = \{(i_1, k_1), \ldots, (i_h, k_h)\}$, and $\sigma = \sum_{t=1}^{h} c(i_t, k_t)$. Then, for any item $(i, k) \notin (S^*_h \cup E(S^*_h))$, we have $c(i, k) \leq \sigma/h$. Let z^*, $\hat{z}$ denote the optimal (integral) solution and the solution output by the scheme for the R-MMCK instance, respectively. Denote by $\mathbf{x}^B(S^*_h)$, $\mathbf{x}^I(S^*_h)$ the basic and integral solutions of LP(S) as computed by the scheme, for the initial guess S^*_h.

By the above rounding method, for any $1 \leq i \leq n$, the cost of the item selected from A^i is $c(i, k_{max})$. Let F denote the set of items for which the basic variable was a fraction, that is, $F = \{(i, k)|\ x^B_{i,k}(S^*_h) < 1\}$, and let $\delta = \sum_{(i,k)\in F} c(i, k)$.

Then, we get that

$$z^* \geq \sum_{i=1}^{n} \sum_{k=1}^{K^i} c(i, k) x^B_{i,k}(S^*_h)$$

$$\geq \sum_{i=1}^{n} \sum_{k=1}^{K^i} c(i, k) x^I_{i,k}(S^*_h) - \delta$$

Recall that in any *basic* solution for an LP, the number of nonzero variables is bounded by the number of tight constraints in some optimal solution. Assume that in the optimal (fractional) solution of $LP(S^*_h)$ there are L tight constraints, where $0 \leq L \leq n + R$. Then in the basic solution $\mathbf{x}^B(S^*_h)$, at most L variables can be strictly positive. Thus, at least $L - 2R$ variables get an integral value (i.e., "1"), and $|F| \leq 2R$. Note that for any $(i, k) \in F$, $c(i, k) \leq \sigma/h$, since $F \cap (S^*_h \cup E(S^*_h)) = \emptyset$. Hence, we get that $z^* \geq \hat{z} + \frac{2R\sigma}{h} \geq \hat{z} + \frac{2R\hat{z}}{h} \geq \frac{\hat{z}}{1+\varepsilon}$. □

The next two lemmas follow from the rounding method used by the scheme.

Lemma 9.5

The scheme yields a feasible solution for the CIP instance.

Lemma 9.6

The cost of the integral solution for the CIP instance is at most $\hat{z} + \varepsilon C$.

Note that C can be guessed in polynomial time within factor $(1 + \varepsilon)$, using binary search over the range $(0, \sum_{i=1}^{n} d_i c_i)$. Thus, combining the above lemmas we get:

Theorem 9.6

There is a PTAS for CIP in fixed dimension.

Consider now the special case where the multiplicity constraints are omitted; that is, each variable x_i can get any nonnegative (integral) value. For this special case, we can use a linear programming formulation in which the number of constraints is R, which is fixed. A PTAS for this problem can be derived from the scheme of Chandra et al. [25] for integer multidimensional knapsack. Drawing from recent results for CIPs, we describe below the PTAS in Ref. [21], which improves the running time in Ref. [25] by using a fast approximation scheme for solving the LP.

A Scheme for CIP$_\infty$

The scheme, called below *multidimensional cover with parameter* ε (MDC_ε), proceeds in the following steps:

(i) For a given $\varepsilon \in (0, 1)$, let $\delta = \lceil R \cdot ((1/\varepsilon) - 1)\rceil$.

(ii) Renumber the items by $1, \ldots, n$, such that $c_1 \geq c_2 \geq \cdots \geq c_n$.

(iii) Denote by Ω the set of integer vectors $\mathbf{x} = (x_1, \ldots, x_n)$ satisfying $x_i \geq 0$, and $\sum_{i=1}^n x_i \leq \delta$. For any vector $\mathbf{x} \in \Omega$: Let $d \geq 1$ be the maximal integer i for which $x_i \neq 0$. Find a $(1 + \varepsilon)$-approximation to the optimal (fractional) solution of the following LP.

$$
\begin{aligned}
(LP') \quad & \text{minimize} && \sum_{i=d+1}^{n} c_i z_i \\
& \text{subject to} && \sum_{i=d+1}^{n} a_{ij} z_i \geq b_j - \sum_{i=1}^{n} a_{ij} x_i \ \text{ for } j = 1, \ldots, R \qquad (9.5)\\
& && z_i \geq 0 \ \text{ for } i = d+1, \ldots, n
\end{aligned}
$$

The constraints (9.5) reflect the fact that we need to fill in each dimension j at least the capacity $b_j - \sum_{i=1}^n a_{ij} x_i$, once we obtained the vector $\mathbf{x}$.

Let $\hat{z}_i$, $d + 1 \leq i \leq n$, be a $(1 + \varepsilon)$-approximate solution for LP'. We take $\lceil \hat{z}_i \rceil$ as the integral solution. Denote by $C_{MDC}(\mathbf{x}) = \sum_{i=d+1}^n c_i \lceil \hat{z}_i \rceil$, the value obtained from the rounded solution, and let $c(\mathbf{x}) = \sum_{i=1}^n c_i x_i$.

(iv) Select the vector $\mathbf{x}^*$ for which $C_{MDC_\varepsilon}(\mathbf{x}^*) = \min_{\mathbf{x}}(c(\mathbf{x}) + C_{MDC}(\mathbf{x}))$.

We now show that MDC_ε is a PTAS for CIP_∞. Let C_o be the cost of an optimal integral solution for the CIP_∞ instance.

Theorem 9.7

MDC_ε is a PTAS for CIP_∞ which satisfies the following. (i) If $C_o \neq 0, \infty$ then $C_{MDC_\varepsilon}/C_o < 1 + \varepsilon$. (ii) The running time of algorithm MDC_ε is $O(n^{\lceil R/\varepsilon \rceil} \cdot \frac{1}{\varepsilon^2} \log C)$, where $C = \max_{1 \leq i \leq n} c_i$ is the maximal cost of any item, and its space complexity is $O(n)$.

To prove the theorem, we need the next lemma.

Lemma 9.7

For any $\varepsilon > 0$, a $(1+\varepsilon)$-approximation to the optimal solution for LP' can be found in $O(1/\varepsilon^2 R \log(C \cdot R))$ steps.

Proof

For a system of inequalities as given in LP', there is a solution in which at most R variables get nonzero values. This follows from the fact that the number of nontrivial constraints is R. Hence, it suffices to solve LP' for the $\binom{n-d}{R}$ possible subsets of R variables, out of $(z_{d+1}, \ldots, z_n)$. This can be done in polynomial time since R is fixed. Now, for each subset of R variables, we have an instance of the *fractional covering problem*, for which we can use a fast approximation scheme (see, e.g., in Ref. [26]) to obtain a $(1 + \varepsilon)$-approximate solution. □

Proof of Theorem 9.7

For showing (i), assume that the optimal (integral) solution for the CIP_∞ instance is obtained by the vector $\mathbf{y} = (y_1, \ldots, y_n)$. If $\sum_{i=1}^n y_i \leq \delta$ then $C_{MDC_\varepsilon} = C_o$, since in this case $\mathbf{y}$ is a valid solution, and $\mathbf{y} \in \Omega$, therefore, in some iteration MDC_ε will examine $\mathbf{y}$. Suppose that $\sum_{i=1}^n y_i > \delta$, then we define the vector $\mathbf{x} = (y_1, \dots, y_{d-1}, x_d, 0, \dots, 0)$, such that $y_1 + \cdots + y_{d-1} + x_d = \delta$. (Note that $x_d \neq 0$.) Let $\tilde{C}_o(\mathbf{x}) = \sum_{i=d+1}^{n} c_i \hat{z}_i$ be the approximate fractional solution for LP'. We have that $\mathbf{x} \in \Omega$, therefore,

$$C_{MDC}(\mathbf{x}) - \tilde{C}_o(\mathbf{x}) \leq Rc_d \tag{9.6}$$

Let $C_o(\mathbf{x})$ be the optimal fractional solution for LP' with the vector $\mathbf{x}$. Note that C_o, the optimal (integral) solution for CIP_∞, satisfies

$$C_o > c(\mathbf{x}) + C_o(\mathbf{x}) \tag{9.7}$$

since $C_o(\mathbf{x})$ is a lower bound for the cost incurred by the integral values $y_{d+1}, \dots, y_n$. In addition,

$$c(\mathbf{x}) + C_{MDC}(\mathbf{x}) \geq C_{MDC_\varepsilon} \tag{9.8}$$

Hence, we get that

$$\begin{aligned}
\frac{C_o}{C_{MDC_\varepsilon}} &\geq \frac{c(\mathbf{x}) + C_o(\mathbf{x})}{c(\mathbf{x}) + C_{MDC}(\mathbf{x})} > 1 - \frac{C_{MDC}(\mathbf{x}) - C_o(\mathbf{x})}{c(\mathbf{x}) + C_{MDC}(\mathbf{x}) - C_o(\mathbf{x})} \\
&\geq 1 - \frac{C_{MDC}(\mathbf{x}) - \tilde{C}_o(\mathbf{x})(1-\varepsilon)}{c(\mathbf{x}) + C_{MDC}(\mathbf{x}) - \tilde{C}_o(\mathbf{x})} \\
&\geq (1-\varepsilon)\left(1 - \frac{C_{MDC}(\mathbf{x}) - \tilde{C}_o(\mathbf{x})}{c(\mathbf{x}) + C_{MDC}(\mathbf{x}) - \tilde{C}_o(\mathbf{x})}\right) \\
&\geq (1-\varepsilon)\left(1 - \frac{C_{MDC}(\mathbf{x}) - \tilde{C}_o(\mathbf{x})}{\delta c_d + C_{MDC}(\mathbf{x}) - \tilde{C}_o(\mathbf{x})}\right)
\end{aligned}$$

The first inequality follows from Eq. (9.7) and Eq. (9.8), and the third inequality follows from the fact that $\tilde{C}_o(\mathbf{x})(1-\varepsilon) \leq C_o(\mathbf{x}) \leq \tilde{C}_o(\mathbf{x})$. The last inequality follows from the fact that $c(\mathbf{x}) \geq \delta c_d$.

Using Eq. (9.6), we get that $\frac{C_o}{C_{MDC_\varepsilon}} \geq (1-\varepsilon)1 - Rc_d/(\delta c_d + Rc_d) \geq (1-\varepsilon)^2$. Taking in the scheme $\tilde{\varepsilon} = \varepsilon/2$, we get the statement in (i).

Next, we show (ii). Note that $|\Omega| = O(n^\delta)$ since the number of possible choices of n nonnegative integers, whose sum is at most δ is bounded by $\binom{n+\delta}{\delta}$. Now, given a vector $\mathbf{x} \in \Omega$, we can compute $C_{MDC}(\mathbf{x})$ in $O(n^R)$ steps since at most R variables out of $z_{d+1}, \dots, z_n$ can have nonzero values. Multiplying by the complexity of the FPTAS for fractional covering, as given in Lemma 9.7, we get the statement of the theorem. □

Enumeration is combined with LP rounding also in the PTAS of Caprara et al. [27] for the *knapsack problem with cardinalities constraints*, and in a PTAS for the *multiple knapsack* problem due to Chekuri and Khanna [28], among others. The scheme in Ref. [27] is based on the scheme of Frieze and Clarke [24], with the running time improved by factor of n, the number of items. The scheme in Ref. [24] is also the basis for PTASs for other variants of the knapsack problem. (A comprehensive survey is given in Ref. [29]; see also Ref. [30].)

9.4 Approximation Schemes for Geometric Problems

In this section we present approximation techniques that are specialized for geometric optimization problems. For a complete description of these techniques we refer the reader to the survey by Arora [31], Chapter 11 in Ref. [32], and Chapter 8 and Section 9.3.3 in Ref. [33]. A typical input for a geometric problem is a set of elements in the space (such as points in the plane); the goal is to connect or pack these elements in a way that minimizes the resources used (e.g., total length of connecting lines, total number of covering objects).

9.4.1 Randomized Dissection

We present below the techniques used in the PTAS of Arora [34] for the Euclidean Traveling Salesman Problem (TSP). In the classical TSP, given are nonnegative edge weights for the complete graph K_n, and the goal is to find a tour of minimum cost, where a tour refers to a cycle of length n. In other words, the goal is to find an ordering of the nodes such that the total cost of the edges along the path visiting all nodes according to this ordering is minimal. In general, TSP is NP-hard in the strong sense, and it cannot be approximated within any multiplicative factor $c > 1$, unless P = NP. The PTAS of Arora considers the relaxed problem of *Euclidean* TSP. The input is a set of n points in $\Re^d$, and the edge weights are the Euclidean (ℓ_2) distances between them.

The idea of the PTAS is to dissect the plane into squares, and to look (using dynamic programming) for a tour that crosses the resulting grid lines only at specific points, denoted *portals*. The parameter ε of the PTAS determines the depth of the recursive dissection as well as the density of the portals. A smaller ε results in more portals and a finer dissection, which lead to a less restricted tour and a larger dynamic programming instance. Randomization is used to determine an initial shift of the grid lines.

A dissection of a square is a recursive partitioning into squares. It can be viewed as a tree of squares whose root is the square we started with. Each square in the tree is partitioned into four equal squares, which are its children. The leaves are squares of a small sidelength—determined by the parameter ε of the PTAS.

The location of the grid lines is determined *randomly* as follows. Given a set of n points in $\Re^2$, enclose the points in a minimum bounding square. Let ℓ be the side of this square. Let $p \in \Re^2$ be the lower left endpoint of the bounding box. Enclose the bounding box inside a larger square, denoted the *enclosing box* of sidelength $L = 2\ell$, and position the enclosing box such that p has distance a from the left edge and b from the lower edge, where $a, b \leq \ell$ are chosen randomly. The randomized dissection is the dissection of this enclosing box. Note that the randomness is used only to determine the placement of the enclosing box (and its accompanying dissection).

We now describe the PTAS in Ref. [34] for the Euclidean TSP, which uses the above randomized dissection. Formally, for every $\varepsilon > 0$, this PTAS finds a $(1 + \varepsilon)$-approximation to Euclidean TSP.

First, perform randomized dissection to the bounding box of the n points. Recall that L is the side of the enclosing box. The recursive procedure of subdividing the squares stops when the sidelengths of the squares becomes less than $L\varepsilon/8n$, or when each square at the last level contains at most one point. We may assume (by scaling) that L is a power of 2 and that the sides of squares at the last level are unit length. Thus, at most $\log L$ iterations are required, and $L \leq 8n/\varepsilon$. When there is more than one point in a unit square, consolidate them into one new "bigger" point. Any tour for the resulting set of points can be augmented to a tour for the original set of points with an increase in length bounded by $\sqrt{2}nL\varepsilon/8n$, which is negligible, since $L \leq OPT/2$. Henceforth, we shall assume that there is at most one point per unit square.

The *level* of a square in the dissection is its depth in the recursive dissection tree; the root square has level 0. We also assign a level from 0 to $\log(L - 1)$ to each horizontal and vertical grid line that participates in the dissection. The horizontal (resp., vertical) line that divides the enclosing box into two has level 0. Similarly, the 2^i horizontal and 2^i vertical lines that divide the level i squares into level $i + 1$ squares have level i. The following property of a randomized dissection is used: Any fixed vertical grid line that intersects the bounding box of the instance has probability $\frac{2^i}{\ell} = \frac{2^{i+1}}{L}$ to be a line at level i.

Next, the location of the portals is determined. Let $m = \frac{1}{\varepsilon} \log L$. The parameter m is the *portal parameter* that determines the density of the points the path can pass through. A level i line has $2^{i+1}m$ equally spaced portals. In addition, we also refer to the corners of each square as a portal. Since a level i line has 2^{i+1} level $i + 1$ squares touching it, it follows that each side of the square has at most $m + 2$ portals (m regular portals plus the 2 corners), and a total of at most $4m + 4$ portals on its boundary. A *portal-respecting tour* is one that, whenever it crosses a grid line, does so at a portal.

Finally, dynamic programming is used to find the optimum portal-respecting tour in time $2^{O(m)} L \log L$. Since $m = O(\log n/\varepsilon)$, we get a total running time of $n^{O(1/\varepsilon)}$. The dynamic programming as well as the complete analysis of bounding the PTAS error and the time complexity are given in Ref. [31].

Note that since the PTAS uses randomization, the error of the PTAS is a random variable. Formally, let OPT denote the cost of the optimum salesman tour and $OPT_{a,b,m}$ denote the cost of the best portal-respecting tour when the portal parameter is m and the random shifts are a, b.

Theorem 9.8

The expectation (over the choices of a, b) of $OPT_{a,b,m} - OPT$ is at most $2 \log L/mOPT$, where L is the sidelength of the enclosing box.

As mentioned in the survey of Arora [31], this method of dissection can be used to develop PTASs for other geometric optimization problems such as minimum Steiner tree, facility location with capacities and demands, and Euclidean min-cost k-connected subgraph.

Another class of geometric optimization problem is the class of *clustering* problems, such as metric max-cut and k-median. In recent research on clustering problems, a core idea in the design of approximation schemes is to use random sampling of data points from a biased distribution, which depends on the pairwise distances. This technique is used, e.g., in the PTAS of Fernandez de la Vega and Kenyon for metric max-cut [35], and in the work of Indyk on metric 2-clustering [36]. For more details on the technique and its applications, we refer the reader to Ref. [37].

9.4.2 Shifted Plane Partitions

The *shifting* technique that is applied to geometric problems is based on selecting the best solution over a (polynomial size) set of feasible solutions. Each candidate feasible solution is obtained using a divide-and-conquer approach, in which the plane is partitioned into disjoint areas (strips). The technique can be applied to geometric problems such as square packing or covering with disks, which arise in Very Large Scale Integration (VLSI) design, image processing, and many other important areas. A common goal in these problems is to cover or pack elements (e.g., points in the plane) into a minimal number of objects (e.g., squares of given size).

Recall that each candidate solution is obtained by using divide-and-conquer approach, in which the plane is partitioned into strips. A solution for the original problem is formed by taking the union of the solutions for these strips. Consecutive solutions refer to consecutive partitions of the plane into strips, which differ from each other by *shifting the partitioning bars*, using the shifting parameter. The smaller the shifting parameter, the larger the number of candidate solutions to be considered, and the better resulting approximation.

We illustrate the shifting technique for the problem of covering n points in the two-dimensional plane. The complete analysis is given in Refs. [33,38]. Assume that the n points are enclosed in an area I. The goal is to cover these points with a minimal number of disks of diameter D. Denote by ℓ the shifting parameter. The area I is divided into vertical strips of width D. Each set of ℓ consecutive strips are grouped together to form strips of width ℓD. Note that there are ℓ different ways to determine this grouping—and they can derive from each other by shifting the partitioning bars to the right over distance D. Denote the ℓ distinct partitions obtained this way by $S_1, S_2, \ldots, S_\ell$.

Let A be an algorithm to solve the covering problem on strips of width at most ℓD. The algorithm A can be used to generate a solution for a given partition S_j. We apply A to each strip in S_j and then union the sets of disks used. The *shift algorithm*, s_A, defined for a given A, uses A to solve the problem for the ℓ possible partitions and selects the solution that requires minimum number of disks.

The following lemma gives the performance ratio of s_A (denoted r_{s_A}) as function of ℓ and the performance ratio of A (denoted r_A).

Lemma 9.8

$$r_{s_A} \leq r_A \left(1 + \frac{1}{\ell}\right)$$

The algorithm A may itself be derived from an application of the shifting technique. In our example, to solve the covering problem on a strip of width ℓD, the strip is cut into squares of size $\ell D \times \ell D$, for which an optimal solution can be found by exhaustive search.

We note that the above shifting technique can be used to derive PTASs for several other problems, including minimum vertex cover and maximum independent set in planar graphs [39]. The idea is that a planar graph can be decomposed into components of bounded outer-planarity. The solution for each component can be found using dynamic programming. The shifting idea is to remove one "layer" from the graph in each iteration. This removal guarantees that the number of cross-cluster edges is small, so by considering the union of the local cluster solutions one can get a good approximation for the original problem.

9.5 Concluding Remarks

There are many other interesting applications of the techniques described in this chapter. We mention a few of them. Golubchik et al. [49] apply enumeration to a *structured* instance in solving the problem of data placement on disks (see also Ref. [40]). The technique of extending solutions for small subsets is applied by Khuller et al. [41] to the problem of broadcasting in heterogeneous networks. Kenyon et al. [42] used a nontrivial combination of grouping with periodic scheduling to obtain a PTAS for data broadcast.

As mentioned in Section 9.4, some techniques are specialized for certain types of problems. For graph problems, some PTASs exploit the density of the input graph (see, e.g., Ref. [43]). There are PTASs which build on the properties of *planar graphs* (see, e.g., Refs. [44,45]).

Finally, we have mentioned in Sections 9.2.3 and 9.4 some techniques used in *randomized* approximation schemes. A detailed exposition of randomized approximation schemes for counting problems is given in Chapter 11 in Ref. [46] (see also Chapter 12 of this handbook). Benczúr and Karger presented in Ref. [47] randomized approximation schemes for cuts and flows in capacitated graphs. Efraimidis and Spirakis used in Ref. [48] the technique of *filtered randomized rounding* in developing randomized approximation schemes for scheduling unrelated parallel machines.

References

[1] Hochbaum, D. S. and Shmoys, D. B., Using dual approximation algorithms for scheduling problems: practical and theoretical results, *JACM*, 34(1), 144, 1987.

[2] Bansal, N. and Sviridenko, M., Two-dimensional bin packing with one dimensional resource augmentation, (submitted for publication).

[3] Arora, S. and Karakostas, G., Approximation schemes for minimum latency problems, *SIAM J. Comput.*, 32(5), 1317, 2003.

[4] Chekuri, C. and Khanna, S., Approximation schemes for preemptive weighted flow time, *Proc. of STOC*, 2002, p. 297.

[5] Khanna, S. and Motwani, R., Towards a syntactic characterization of PTAS, *Proc. of STOC*, 1996, p. 329.

[6] Woeginger, G. J., There is no asymptotic PTAS for two-dimensional vector packing, *Inf. Process. Lett.*, 64, 293, 1997.

[7] Motwani, R., Lecture Notes on Approximation Algorithms, Technical report, Department of Computer Science, Stanford University, CA, 1992.

[8] Schuurman, P. and Woeginger, G. J., Approximation schemes—a tutorial, in *Lectures on Scheduling*, Möehring, R. H., Potts, C. N., Schulz, A. S., Woeginger, G. J., and Wolsey, L. A., Eds.

[9] Karger, D., Stein, C., and Wein, J., Scheduling algorithms, in *Handbook of Algorithms and Theory of Computation*, Atallah, M. J., Ed., CRC Press, Boca Raton, FL, 1997.

[10] Graham, R. L., Bounds for certain multiprocessing anomalies, *Bell Syst. Tech. J.*, 45, 1563, 1966.

[11] Sahni, S., Approximate algorithms for the 0/1 knapsack problem, *JACM*, 22, 115, 1975.

[12] Sahni, S., Algorithms for scheduling independent tasks, *JACM*, 23, 555, 1976.

[13] Sevastianov, S. V. and Woeginger, G. J., Makespan minimization in open shops: a polynomial time approximation scheme, *Math. Program.*, 82, 191, 1998.

[14] Jansen, K. and Sviridenko, M., Polynomial time approximation schemes for the multiprocessor open and flow shop scheduling problem, *Proc. of STACS*, 2000, 455.

[15] Afrati, F. N., Bampis, E., Chekuri, C., Karger, D. R., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., and Sviridenko, M., Approximation schemes for minimizing average weighted completion time with release dates, *Proc. of FOCS*, 44, 1999, 32.

[16] Shachnai, H., Tamir, T., and Yehezkely, O., Approximation schemes for packing with item fragmentation, *3rd Workshop on Approximation and Online Algorithms (WAOA)*, Palma de Mallorca, Spain, 2005.

[17] Chekuri, C. and Khanna, S., On multidimensional packing problems, *SIAM J. Comput.*, 33(4), 837, 2004.

[18] Potts, C. N., Analysis of a linear programming heuristic for scheduling unrelated parallel machines, *Discrete Appl. Math.*, 10, 155, 1985.

[19] Jansen, K., Solis-Oba, R., and Sviridenko, M., Makespan minimization in job shops: a linear time approximation scheme, *SIAM J. Discrete Math.*, 16(2), 288, 2003.

[20] Garey, M. R. and Graham, R. L., Bounds for multiprocessor scheduling with resource constraints, *SIAM J. Comput.*, 4(2), 187, 1975.

[21] Shachnai, H., Shmueli, O., and Sayegh, R., Approximation schemes for deal splitting and covering integer programs with multiplicity constraints, *Proc. of WAOA*, 2004, p. 111.

[22] Kolliopoulos, S. G., Approximating covering integer programs with multiplicity constraints, *Discrete Appl. Math.*, 129(2–3), 461, 2003.

[23] Kolliopoulos, S. G. and Young, N. E., Tight approximation results for general covering integer programs, *Proc. of FOCS*, 2001, p. 522.

[24] Frieze, A. M. and Clarke, M. R. B., Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses, *Eur. J. Oper. Res.*, 15(1), 100, 1984.

[25] Chandra, A. K., Hirschberg, D. S., and Wong, C. K., Approximate algorithms for some generalized knapsack problems, *Theor. Comput. Sci.*, 3, 293, 1976.

[26] Fleischer, L., A fast approximation scheme for fractional covering problems with variable upper bounds, *Proc. of SODA*, 2004, p. 994.

[27] Caprara, A., Kellerer, H., Pferschy, U., and Pisinger, D., Approximation algorithms for knapsack problems with cardinality constraints, *Eur. J. Oper. Res.*, 123, 333, 2000.

[28] Chekuri, C. and Khanna, S., A PTAS for the multiple knapsack problem, *Proc. of SODA*, 2000, p. 213.

[29] Kellerer, H., Pferschy, U., and Pisinger, D., *Knapsack Problems*, Springer, Berlin, 2004.

[30] Shachnai, H. and Tamir, T., Approximation schemes for generalized 2-dimensional vector packing with application to data placement, *Proc. of APPROX*, 2003.

[31] Arora, S., Approximation schemes for NP-hard geometric optimization problems: a survey, *Math. Program.*, Springer, Berlin/Heidelberg, 97(1–2), 43–69, 2003.

[32] Vazirani, V. V., *Approximation Algorithms,* Springer, Berlin, 2001.

[33] Hochbaum, D. S., *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, Boston, MA, 1995.

[34] Arora, S., Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems, *JACM*, 45(5), 753, 1998.

[35] Fernandez de la Vega, W. and C. Kenyon, C., A randomized approximation scheme for metric MAX-CUT, *Proc. of FOCS,* 1998, p. 468.

[36] Indyk, P., A sublinear time approximation scheme for clustering in metric spaces, *Proc. of FOCS*, 1999.

[37] Fernandez de la Vega, W., Karpinski, M., Kenyon, C., and Rabani, Y., Approximation schemes for clustering problems, *Proc. of STOC*, 2003.

[38] Hochbaum, D. S. and Maass, W., Approximation schemes for covering and packing problems in image processing and VLSI, *JACM*, 32(1), 130, 1985.

[39] Baker, B. S., Approximation algorithms for NP-complete problems on planar graphs, *JACM*, 41(1), 153, 1994.

[40] Kashyap, S. and Khuller, S., Algorithms for non-uniform size data placement on parallel disks, *Proc. of FST & TCS*, Lecture Notes in Computer Science, Vol. 2914, Springer, Berlin, 2003.

[41] Khuller, S., Kim, Y., and Woeginger, G., A polynomial time approximation scheme for broadcasting in heterogeneous networks, *Proc. of APPROX*, 2004.

[42] Kenyon, C., Schabanel, N., and Young, N. E., Polynomial-time approximation scheme for data broadcast, *CoRR cs.DS/0205012*, 2002.

[43] Arora, S., Karger D., and Karpinski, M., Polynomial time approximation schemes for dense instances of NP-hard problems, *Proc. of STOC*, 1995.

[44] Halldórsson, M. M. and Kortsarz, G., Tools for multicoloring with applications to planar graphs and partial k-trees, *J. Algorithms*, 42(2), 334, 2002.

[45] Demaine, E. D. and Hajiaghayi, M., Bidimensionality: new connections between FPT algorithms and PTASs, *Proc. of SODA*, 2005, p. 590.

[46] Motwani, R. and Raghavan, P., *Randomized Algorithms,* Cambridge University Press, Cambridge, 1995.

[47] Benczúr, A. A. and Karger, D. R., Randomized approximation schemes for cuts and flows in capacitated graphs, Technical report, MIT, July 2002.

[48] Efraimidis, P. and Spirakis, P. G., Randomized approximation schemes for scheduling unrelated parallel machines, *Electronic Colloquium on Computational Complexity (ECCC)*, 7(7), 2000.

[49] Golubchik, L., Khanna, S., Khuller, S., Thurimella, R., and Zhu, A., Approximation algorithms for data placement on parallel disks. In *Proc. of (SODA)*, 223–232, 2000.

10

Rounding, Interval Partitioning, and Separation

Sartaj Sahni
University of Florida

10.1 Introduction

This chapter reviews three general methods—rounding, interval partitioning, and separation—proposed by Sahni [1] to transform pseudopolynomial-time algorithms into fully polynomial-time approximation schemes. The three methods, which generally apply to dynamic-programming and enumeration-type pseudopolynomial-time algorithms, are illustrated using the 0/1-knapsack and multiconstrained shortest paths problems. Both of these problems are known to be NP-hard and both are solvable in pseudopolynomial time using either dynamic programming or enumeration.

10.2 Rounding

The rounding method of Ref. [1] is also known by the names digit truncation and scaling. The key idea in the rounding method is to reduce the magnitude of some or all of the numbers in an instance so that the pseudopolynomial-time algorithm actually runs in polynomial time on the reduced instance. The amount by which each number is reduced is such that the optimal solution for the reduced instance is an ϵ-approximate solution for the original instance.

Rounding up, rounding down, and random rounding are three possible strategies to construct the reduced instance. In each, we employ a rounding factor $\delta(n, \epsilon)$, where n is a measure of the problem size. For convenience, we abbreviate $\delta(n, \epsilon)$ as δ. When rounding up, each number α (for convenience, we assume that all numbers in all instances are positive) that is to be rounded is replaced by $\lceil \alpha/\delta \rceil$ and when rounding down, α is replaced by $\lfloor \alpha/\delta \rfloor$. In random rounding, we round up with probability equal to the fractional part of α/δ and round down with probability equal to 1—the fractional part of α/δ. So, for example, if $\alpha = 7$ and $\delta = 4$, α is replaced by (or reduced to) 2 when rounding up and by 1 when rounding down. In random rounding, α is replaced by 2 with probability 0.75 and by 1 with probability 0.25. Random rounding is typically implemented using a uniform random number generator that generates real numbers in the range [0, 1). The decision on whether to round up or down is made by generating a random number. If the generated number is $\leq$ the fractional part of α/δ, we round up; otherwise, we round down.[1]

As an example of the application of rounding, consider the 0/1-knapsack problem, which is known to be NP-hard [3]. In the 0/1-knapsack problem, we wish to pack a knapsack (bag or sack) with a capacity of c. From a list of n items/objects, we must select the items that are to be packed into the knapsack. Each item i has a weight w_i and a profit p_i. We assume that all weights and profits are positive integers. In a feasible knapsack packing, the sum of the weights of the packed objects does not exceed the knapsack capacity c, which also is assumed to be a positive integer. Since an item with weight more than c cannot be in any feasible packing, we may assume that $w_i \leq c$ for all i. An optimal packing is a feasible packing with maximum profit. The problem formulation is

$$\text{maximize} \quad \sum_{i=1}^{n} p_i x_i$$

subject to the constraints

$$\sum_{i=1}^{n} w_i x_i \leq c \text{ and } x_i \in \{0, 1\}, \quad 1 \leq i \leq n$$

In this formulation we are to find the values of x_i. When $x_i = 1$, it means that object i is packed into the knapsack, and $x_i = 0$ means that object i is not packed.

For the instance $n = 5$, $(p_1, \ldots, p_5) = (w_1, \ldots, w_5) = \{1, 2, 4, 8, 16\}$ and $c = 27$, the optimal solution is $X = (x_1, x_2, \ldots, x_5) = (1, 1, 0, 1, 1)$, which corresponds to packing items 1, 2, 4, and 5 into the knapsack. This solution uses all of the knapsack capacity and yields a profit of 27. With each feasible packing, we associate a profit and weight pair (P, W), where P is the sum of the profits of the items in the packing and $W \leq c$ the sum of their weights. For example, a packing that generates a profit of 15 and uses 20 units of capacity is represented by the pair (15, 20). P is the profit or value of the packing (P, W) and W its weight.

Several of the standard algorithm design methods of Ref. [3]—for example backtracking, branch and bound, dynamic programming, and divide and conquer—may be applied to the knapsack problem. Backtracking and branch and bound result in algorithms whose complexity is $O(2^n)$ and dynamic programming results in a pseudopolynomial-time algorithm whose complexity is $O(\min\{2^n, n\tilde{F}, nc\})$, where $\tilde{F}$ is the value of the optimal solution [4]. A pseudopolynomial-time algorithm with this same complexity also may be arrived at using an enumerative approach. By coupling a divide and conquer step to this enumerative algorithm, we obtain a pseudopolynomial-time algorithm whose complexity is $O(\min\{2^{n/2}, n\tilde{F}, nc\})$ [4].

Let $(P1, W1)$ and $(P2, W2)$ represent two different feasible packings of items selected from the first i items. Tuple $(P1, W1)$ *dominates* $(P2, W2)$ iff either $P1 \geq P2$ and $W1 < W2$ or $P1 > P2$ and $W1 = W2$. The enumerative algorithm for the 0/1-knapsack problem constructs a list of (or enumerates) the profit and weight pairs that correspond to all possible nondominated feasible packings. This list is constructed incrementally. Let S_i be the list of nondominated profit and weight pairs for all possible feasible packings chosen from the first i items. We start with the list $S_0 = \{(0, 0)\}$, and construct $S_1, S_2, \ldots, S_n$ in this order. Note that each S_i, $i > 0$, may be constructed from S_{i-1} using the equality

$$S_i = S_{i-1} \oplus \{(a + p_i, b + w_i) | (a, b) \in S_{i-1} \text{ and } b + w_i \leq c\} \tag{10.1}$$

where $\oplus$ denotes a union in which dominated pairs are eliminated. Eq. (10.1) simply states that the nondominated pairs obtainable from the first i items are a subset of those obtainable from the first $i - 1$

[1] There is a similar sounding, but quite different, method for approximation algorithms—randomized rounding —due to Raghavan and Thompson [2]. In randomized rounding, we start with an integer linear program formulation; relax the integer constraints to real number constraints; solve the resulting linear program; transform the noninteger values in the obtained solution to the linear program to integers using the random rounding strategy stated above.

items (these have $x_i = 0$) plus those obtainable from feasible packings of the first i items that necessarily include w_i (i.e., $x_i = 1$). The subset is identified by eliminating dominated pairs. Trying Eq. (10.1) out in the above $n = 5$ instance, we get (since $P = W$ for every pair (P, W) in this example, we represent each pair by a single number)

$$\begin{aligned}
S_0 &= \{0\}\\
S_1 &= \{0\} \oplus \{1\} = \{0, 1\}\\
S_2 &= \{0, 1\} \oplus \{2, 3\} = \{0, 1, 2, 3\}\\
S_3 &= \{0, 1, 2, 3\} \oplus \{4, 5, 6, 7\} = \{0, 1, 2, 3, 4, 5, 6, 7\}\\
S_4 &= \{0, \ldots, 7\} \oplus \{8, \ldots, 15\} = \{0, \ldots, 15\}\\
S_5 &= \{0, \ldots, 15\} \oplus \{16, \ldots, 27\} = \{0, \ldots, 27\}
\end{aligned}$$

For the case $n = 4$, $(p_1, \ldots, p_4) = (w_1, \ldots, w_4) = \{1, 1, 8, 8\}$, and $c = 17$, we get

$$\begin{aligned}
S_0 &= \{0\}\\
S_1 &= \{0\} \oplus \{1\} = \{0, 1\}\\
S_2 &= \{0, 1\} \oplus \{1, 2\} = \{0, 1, 2\}\\
S_3 &= \{0, 1, 2\} \oplus \{8, 9, 10\} = \{0, 1, 2, 8, 9, 10\}\\
S_4 &= \{0, 1, 2, 8, 9, 10\} \oplus \{8, 9, 10, 16, 17\} = \{0, 1, 2, 8, 9, 10, 16, 17\}
\end{aligned}$$

The solution to the knapsack instance may be determined from the S_is using the procedure of Figure 10.1.

For our $n = 5$ instance with $c = 27$, (P, W) is determined to be $(27, 27)$ in Step 1. In Step 2, x_5 is set to 1 as $(27, 27) \notin S_4$ and P and W are updated to 11. Then x_4 is set to 1 as $(11, 11) \notin S_3$ and P and W are updated to 3. Next, x_3 is set to 0 as $3 \in S_2$. x_2 and x_1 are set to 1 in the remaining two iterations of the **for** loop.

The S_is may be implemented as sorted linear lists (note that the dominance rule ensures that if S_i is in ascending order of P, S_i is also in ascending order of W; also, no two pairs of S_i may have the same P or the same W value). The set S_i may be computed from S_{i-1} in $O(|S_{i-1}|)$ time using Eq. (10.1). The time to compute all S_is is, therefore, $\sum_{1 \le i \le n} |S_{i-1}|$. (Note that in S_n we need to only compute the pair with maximum profit. When the S_is are in ascending order of profit, this maximum pair may be determined easily.) From Eq. (10.1) it follows that $|S_i| \le 2^i$ (this also follows from the observation that there are 2^i different subsets of i items). Also, since the w_is and p_is are positive integers and S_i has only nondominated pairs, $|S_i| \le \min\{\tilde{F}, c\} + 1$. Hence, the time needed to generate the S_is is $O(\min\{2^n, n\tilde{F}, nc\})$. If the sorted linear lists are array lists, each S_i may be searched for (P, W) in $O(\log |S_i|)$ time. In this case the complexity of the procedure to determine the x_is from the S_is is $O(n * \min\{n, \log\ \tilde{F}, \log\ c\})$. This may be reduced to $O(n)$ by retaining with each $(P, W) \in S_i$ a pointer to the pair (P, W) or $(P - p_i, W - w_i)$ that is in S_{i-1} (note that at least one of these pairs must be in S_{i-1}). These pointers are added to the members

```
Step 1: [Determine solution value]
    Determine the pair $(P, W) \in S_n$ with maximum profit value. The value of an optimal packing is $P$.

Step 2: [Determine $x_i$s]
    for $(i = n; i > 0; i - -)$
        if $((P, W) \notin S_{i-1})$ { $x_i = 1$; $P- = p_i$; $W- = w_i$; }
        else $x_i = 0$;
```

FIGURE 10.1 Procedure to determine x_is from the S_is.

of S_i at the time S_i is constructed using Eq. (10.1). The inclusion of these pointers does not change the asymptotic complexity of the procedure to compute the S_is.

The enumerative pseudopolynomial-time algorithm just described for the knapsack problem may be transformed into a fully polynomial-time approximation scheme by suitably rounding down the p_is. Suppose we round using the rounding factor δ to obtain the reduced instance with $p'_i = \lfloor p_i/\delta \rfloor$ and $w'_i = w_i$, $1 \leq i \leq n$, and $c' = c$. The time to solve the reduced instance is $O(n\tilde{F}')$, where $\tilde{F}'$ is the value of the optimal solution to the reduced problem (we assume the reduction is sufficient so that $n\tilde{F}' < \min\{2^n, nc'\}$). Notice that the original and reduced instances have the same feasible packings; only the profit associated with each feasible packing is different. A feasible packing has a smaller profit in the reduced instance than in the original instance.

Consider any feasible packing $(x_1, \ldots, x_n)$. Since $p'_i * \delta \leq p_i < (p'_i + 1) * \delta$,

$$\delta * \sum_i p'_i x_i \leq \sum_i p_i x_i < \delta * \sum_i (p'_i + 1)x_i \tag{10.2}$$

So,

$$\delta \tilde{F}' \leq \tilde{F} < \delta(\tilde{F}' + n) \tag{10.3}$$

Suppose we use the just described rounding strategy on our $n = 4$ example with $(p_1, \ldots, p_4) = (w_1, \ldots, w_4) = (1, 1, 8, 8)$, $c = 17$ and $\delta = 3$. We obtain $(p'_1, \ldots, p'_4) = (0, 0, 2, 2)$, $(w'_1, \ldots, w'_4) = (1, 1, 8, 8)$ and $c' = 17$. One of the optimal solutions for the reduced instance has $(x_1, x_2, x_3, x_4) = (0, 0, 1, 1)$ and the value of this solution is $p'_3 + p'_4 = 4$. In the original instance, the solution (0, 0, 1, 1) has value 16. Note that many different knapsack instances round to the same reduced instance. For example, $(p_1, \ldots, p_4) = (2, 1, 6, 7)$, $(w_1, \ldots, w_4) = (1, 1, 8, 8)$ and $c = 17$ (using $\delta = 3$). The value of the solution (0, 0, 1, 1), for this original instance, is 13. From Eq. (10.2), regardless of the original instance, the value of (0, 0, 1, 1) must be at least $\delta * \sum p'_i x_i = 12$ and cannot equal or exceed $\delta * \sum (p'_i + 1)x_i = 18$.

To ensure that every optimal solution to the reduced instance also defines an ϵ-approximate solution for the original instance, we must select δ carefully. Let $\hat{F}$ be the value, in the original instance, of the optimal solution for the reduced instance. From Eq. (10.2) and Eq. (10.3), we obtain

$$\hat{F} \geq \delta \tilde{F}' > \tilde{F} - n\delta$$

So, $(\tilde{F} - \hat{F}) < n\delta$ and $(\tilde{F} - \hat{F})/\tilde{F} < n\delta/\tilde{F}$. To gurantee that the optimal solution for the reduced instance is an ϵ-approximate solution for the original instance, we require $n\delta/\tilde{F} \leq \epsilon$ or $\delta \leq \epsilon\tilde{F}/n$. Since the reduced instance has smaller p'_i values and hence smaller complexity when δ is larger, we would like to use

$$\delta = \epsilon\tilde{F}/n$$

With this choice of δ, $\tilde{F}' \leq \tilde{F}/\delta = n/\epsilon$ (Eq. [10.3]). So, $|S_i| \leq n/\epsilon + 1$ and the complexity of the enumerative algorithm becomes $O(n^2/\epsilon)$. In other words, the enumerative algorithm becomes a fully polynomial-time approximation scheme for the 0/1-knapsack problem! Unfortunately, this choice of δ is problematic as we cannot easily compute $\tilde{F}$. Since any $\delta \leq \epsilon\tilde{F}/n$ guarantees ϵ-approximate solutions, we may use

$$\delta = \epsilon LB/n$$

where $LB \leq \tilde{F}$ is a lower bound on the value of the optimal solution. Let $Pmax = \max_i\{p_i\}$ be the maximum profit value. Since $w_i \leq c$ for all i (by assumption), $Pmax \leq \tilde{F}$, and $LB = Pmax$ is a lower bound on $\tilde{F}$. So, using $\delta = \epsilon Pmax/n$ guarantees ϵ-approximate solutions. Since $\tilde{F} \leq nPmax$, $\tilde{F}' \leq nPmax/\delta = n^2/\epsilon$ and the complexity of the enumerative algorithm becomes $O(n^3/\epsilon)$.

An alternative way to determine a lower bound for $\tilde{F}$ is to sort the knapsack items into nondecreasing order of profit denisty p_i/w_i and pack the items into the knapsack in density order upto and including the first item that causes the knapsack to be either filled or overfilled. Note that if there is no such first item, all items can be packed into the knapsack and this packing represents the optimal solution. Also note that if the stated packing strategy fills the knapsack completely, it represents an optimal packing. So, assume that the capacity is exceeded. Let $\bar{F}$ be the value of this packing that overfills the knapsack. In Ref. [5], it is shown that $\bar{F}/2 \leq \tilde{F} \leq \bar{F}$. So, using $\delta = \epsilon\bar{F}/(2n)$ as the rounding factor, guarantees ϵ-approximate solutions. Since $\tilde{F} \leq \bar{F}$, $\tilde{F}' \leq \bar{F}/\delta$ and, for the reduced instance, $|S_i| \leq \bar{F}/\delta + 1 = 2n/\epsilon + 1$. For the reduced instance, the complexity of the enumerative algorithm is, therefore, $O(n^2/\epsilon)$ and we get a fully polynomial-time ϵ-approximation scheme of this complexity.

10.3 Interval Partitioning

Unlike rounding, which reduces an instance to one that is easier to solve using a known pseudopolynomial-time algorithm, in interval partitioning we work with the nonreduced (original) instance. In interval partitioning, we partition the solution space into buckets or intervals and for each interval, we retain only one of the (feasible) solutions (or partial solutions) that fall into it.

For the 0/1-knapsack problem, for example, each pair $(P, W) \in S_i, i \leq n$, represents a feasible solution. We may partition the solution space based on the profit value of the pair (P, W). If we partition using an interval size of δ, then the intervals are $[0, \delta)$, $[\delta, 2\delta)$, $[2\delta, 3\delta)$, and so on. When two or more solutions fall into the same interval, all but one of them is eliminated. Specifically, we eliminate all but the one with least weight. Let S_i' be the list of (P, W) pairs for all possible feasible packings chosen from the first i items subject to the interval partitioning constraint that S_i' has at most 1 (P, W) pair in each interval. We begin with $S_0' = \{(0, 0)\}$ and compute S_i' from S_{i-1}' using the equation

$$S_i' = S_{i-1}' \odot \{(a + p_i, b + w_i) | (a, b) \in S_{i-1}' \text{ and } b + w_i \leq c\} \tag{10.4}$$

where $\odot$ denotes a union in which only the least weight pair from each interval is retained. The maximum profit pair in S_n' is used as the approximate optimal solution. The x_is for this pair are obtained using the procedure of Figure 10.1 with S_i replaced by S_i'.

Consider the 0/1-knapsack instance $n = 5, (p_1, \ldots, p_5) = (w_1, \ldots, w_5) = \{1, 2, 4, 8, 16\}$, and $c = 27$, which was first considered in Section 10.2. Suppose we work with an interval size $\delta = 2$. The intervals are $[0, 2)$, $[2, 4)$, $[4, 6)$, and so on. The S_i's are

$$\begin{aligned}
S_0' &= \{0\}\\
S_1' &= \{0\} \odot \{1\} = \{0\}\\
S_2' &= \{0\} \odot \{2\} = \{0, 2\}\\
S_3' &= \{0, 2\} \odot \{4, 6\} = \{0, 2, 4, 6\}\\
S_4' &= \{0, 2, 4, 6\} \odot \{8, 10, 12, 14\} = \{0, 2, 4, 6, 8, 10, 12, 14\}\\
S_5' &= \{0, 2, 4, \ldots, 14\} \odot \{16, 18, 20, \ldots, 26\} = \{0, 2, 4, \ldots, 26\}
\end{aligned}$$

The maximum profit pair in S_4 is $(26, 26)$. For this instance, therefore, the best solution found using interval partitioning with $\delta = 2$ has a profit 1 less than that of the optimal.

Consider the instance $n = 6$, $(p_1, \ldots, p_6) = (w_1, \ldots, w_6) = (1, 2, 5, 6, 8, 9)$, and $c = 27$. Suppose we use $\delta = 3$. The intervals are $[0, 3)$, $[3, 6)$, $[6, 9)$, and so on. The S_i's are

$$\begin{aligned}
S_0' &= \{0\}\\
S_1' &= \{0\} \odot \{1\} = \{0\}\\
S_2' &= \{0\} \odot \{2\} = \{0\}\\
S_3' &= \{0\} \odot \{5\} = \{0, 5\}\\
S_4' &= \{0, 5\} \odot \{6, 11\} = \{0, 5, 6, 11\}\\
S_5' &= \{0, 5, 6, 11\} \odot \{8, 13, 14, 19\} = \{0, 5, 6, 11, 13, 19\}\\
S_6' &= \{0, 5, 6, 11, 13, 19\} \odot \{9, 14, 15, 20, 22\} = \{0, 5, 6, 9, 13, 15, 19, 22\}
\end{aligned}$$

The profit of the best solution found for this instance is 22; the profit for the optimal solution is 27. Note that if c were 28 instead of 27,

$$S'_6 = \{0, 5, 6, 11, 13, 19\} \odot \{9, 14, 15, 20, 22, 28\} = \{0, 5, 6, 9, 13, 15, 19, 22, 28\}$$

and we would have found the optimal solution.

Let $\check{F}$ be the value of the solution found by interval partitioning. It is easy to see that $\tilde{F} < \check{F} + n\delta$. So, $(\tilde{F} - \check{F})/\tilde{F} < n\delta/\tilde{F}$. To guarantee that the solution found using interval partitioning is an ϵ-approximate solution, we require $n\delta/\tilde{F} \le \epsilon$. For this, we must choose δ so that $\delta \le \epsilon\tilde{F}/n$. Since $\tilde{F}$ is hard to compute, we opt to select δ as in Section 10.2. Both the choices $\delta = \epsilon Pmax/n$ and $\delta = \epsilon\bar{F}/(2n)$ guarantee that the solution generated using interval partitioning is ϵ-approximate. When $\delta = \epsilon Pmax/n$, the number of intervals is $\tilde{F}/\delta + 1 \le nPmax/\delta + 1 = n^2/\epsilon + 1$ and the complexity of the (modified) enumerative algorithm is $O(n^3/\epsilon)$. When $\delta = \epsilon\bar{F}/(2n)$, the number of intervals is $\tilde{F}/\delta + 1 \le \bar{F}/\delta + 1 = 2n/\epsilon + 1$ and the complexity is $O(n^2/\epsilon)$.

10.4 Separation

An examination of our $n = 6$ example of Section 10.3 reveals that interval partitioning misses some opportunities to reduce the size of an S'_i while yet preserving the relationship $\tilde{F} \le \hat{F} + n\delta$, which is necessary to ensure an ϵ-approximate solution. For example, in S'_4 we have two solutions, one with value 5 and the other with value 6. Although these are within δ of each other, they fall into two different intervals and so neither is eliminated. In the separation method, we ensure that the value of retained solutions differs by more than δ.

For the 0/1-knapsack problem, let S''_i be the list of (P, W) pairs for all possible feasible packings chosen from the first i items subject to the separation constraint that no two pairs of S''_i have value within δ of each other. We begin with $S''_0 = \{(0, 0)\}$ and compute S''_i from S''_{i-1} using the equation

$$S''_i = S''_{i-1} \otimes \{(a + p_i, b + w_i) | (a, b) \in S''_{i-1} \quad \text{and} \quad b + w_i \le c\} \tag{10.5}$$

where $\otimes$ denotes a union that implements the separation constraint. More precisely, suppose that

$$T = S''_{i-1} \oplus \{(a + p_i, b + w_i) | (a, b) \in S''_{i-1} \quad \text{and} \quad b + w_i \le c\}$$

Let (P_i, W_i), $1 \le i \le |T|$ be the pairs in T in ascending order of profit (and hence of weight). The set S''_i is obtained from T using the code of Figure 10.2.

The maximum profit pair in S''_n is used as the approximate optimal solution.

Consider the $n = 6$ example of Section 10.3. $S''_i = S'_i$, $0 \le i \le 3$. The remaining S''_i's are

$$\begin{aligned}
S''_0 &= \{0\} \\
S''_1 &= \{0\} \otimes \{1\} = \{0\} \\
S''_2 &= \{0\} \otimes \{2\} = \{0\} \\
S''_3 &= \{0\} \otimes \{5\} = \{0, 5\} \\
S''_4 &= \{0, 5\} \otimes \{6, 11\} = \{0, 5, 11\} \\
S''_5 &= \{0, 5, 11\} \otimes \{8, 13, 19\} = \{0, 5, 11, 19\} \\
S''_6 &= \{0, 5, 11, 19\} \otimes \{9, 14, 20\} = \{0, 5, 9, 14, 19\}
\end{aligned}$$

```
$S''_i = \{(P_1, W_1)\}$;
$P_{prev} = P_1$;
for (int $i = 1$; $i <= |T|$; $i++$)
    if ($P_i > P_{prev} + \delta$) {$S''_i = S''_i \cup \{(P_i, W_i)\}$; $P_{prev} = P_i$;}
```

FIGURE 10.2 Computing S''_i from T.

The profit of the best solution found for this instance is 19; the profit for the optimal solution is 27. We could have produced a slightly better solution by noting that we can replace the computation of S''_n by a step in which we determine the maximum profit pair in

$$S''_{n-1} \oplus \{(a + p_i, b + w_i) | (a, b) \in S''_{n-1} \quad \text{and} \quad b + w_i \le c\}$$

For our example, this pair has value 20.

Let $\check{F}$ be the value of the solution found by the separation method. It is easy to see that $\tilde{F} \le \check{F} + n\delta$. So, $\delta \le \epsilon \tilde{F}/n$ ensures that an ϵ-approximate solution is found. As was the case in Section 10.3, for the knapsack problem, the choices $\delta = \epsilon Pmax/n$ and $\epsilon \bar{F}/(2n)$ guarantee that the solution generated using separation is ϵ-approximate. When $\delta = \epsilon Pmax/n$, the complexity of the (modified) enumerative algorithm is $O(n^3/\epsilon)$ and when $\delta = \epsilon \bar{F}/(2n)$, the complexity is $O(n^2/\epsilon)$.

Intuitively, we may expect that using the same δ value, $|S''_i| \le |S'_i|$ for all i. Although this relationship holds for the $n = 6$ example considered above, the relationship does not always hold. For example, consider the knapsack instance $n = 5$, $(p_1, \ldots, p_5) = (w_1, \ldots, w_5) = (30, 10, 51, 51, 51)$, $c = 186$, and $\delta = 20$. Using interval partitioning, we get

$$\begin{aligned}
S'_0 &= \{0\} \\
S'_1 &= \{0\} \odot \{30\} = \{0, 30\} \\
S'_2 &= \{0, 30\} \odot \{10, 40\} = \{0, 30, 40\} \\
S'_3 &= \{0, 30, 40\} \odot \{51, 81, 91\} = \{0, 30, 40, 81\} \\
S'_4 &= \{0, 30, 40, 81\} \odot \{51, 81, 91, 132\} = \{0, 30, 40, 81, 132\} \\
S'_5 &= \{0, 30, 40, 81, 132\} \odot \{51, 81, 91, 132, 183\} = \{0, 30, 40, 81, 132, 183\}
\end{aligned}$$

and using separation, we get

$$\begin{aligned}
S''_0 &= \{0\} \\
S''_1 &= \{0\} \otimes \{30\} = \{0, 30\} \\
S''_2 &= \{0, 30\} \otimes \{10, 40\} = \{0, 30\} \\
S''_3 &= \{0, 30\} \otimes \{51, 81\} = \{0, 30, 51, 81\} \\
S''_4 &= \{0, 30, 51, 81\} \otimes \{51, 81, 102, 132\} = \{0, 30, 51, 81, 102, 132\} \\
S''_5 &= \{0, 30, 51, 81, 102, 132\} \otimes \{51, 81, 102, 132, 153, 183\} = \{0, 30, 51, 81, 102, 132, 153, 183\}
\end{aligned}$$

10.5 0/1-Knapsack Problem Revisited

In Sections 10.2–10.4, we saw how to apply the generic rounding, interval partitioning, and separation methods to the 0/1-knapsack problem and obtain an ϵ-approximate fully polynomial-time approximation scheme for this problem. The complexity of the approximation scheme is either $O(n^3/\epsilon)$ or $O(n^2/\epsilon)$, depending on the choice of δ. By tailoring the approximation method to the application, we can, at times, reduce the complexity of the approximation scheme. Ibarra and Kim [5], for example, combine rounding and interval partitioning to arrive at an $O(n \log n - (\log \epsilon)/\epsilon^4)$ ϵ-approximate fully polynomial-time approximation scheme for the 0/1-knapsack problem. Figure 10.3 gives their algorithm. The correctness proof and complexity analysis can be found in Ref. [5].

10.6 Multiconstrained Shortest Paths

10.6.1 Notation

Assume that a communication network is represented by a weighted directed graph $G = (V, E)$, where V is the set of network vertices or nodes and E the set of network links or edges. We use n and e, respectively,

```
Step 1: [Determine $\delta$]
    Sort the $n$ items into nondecreasing order of profit density $p_i/w_i$.
    Let $\bar{F}$ be as in Section ??.
    Let $\delta = \epsilon^2\bar{F}/9$.
Step 2: [Partition Items]
    Let small be the items with $p_i \leq \epsilon\bar{F}/3$.
    Let big be the remaining items.
Step 3: [Rounding]
    Let big' be obtained from big by rounding down the profits using the rounding factor $\delta$.
    For each rounded-down profit $p$, retain up to $9/(\epsilon^2 p)$ items of least weight.
    Let big'' be the resulting item set.
    Let $m$ be the number of items in big''.
Step 4: [Interval Partitioning]
    Use interval partitioning on big'' and determine $S'_m$.
Step 5: [Augmentation]
    Augment each $(P, W) \in S'_m$ by adding in items from small in order of nondecreasing density so as not
    to exceed the capacity of the knapsack.
    Select the augmentation that yields maximum profit as the approximate solution.
```

FIGURE 10.3 Fully polynomial-time ϵ-approximation scheme of Ref. [5].

to denote the number of nodes and links in the network, that is, $n = |V|$ and $e = |E|$. We assume that each link (u, v) of the network has $k > 1$ nonnegative weights $w_i(u, v)$, $1 \leq i \leq k$. These weights, for example, may represent link cost, delay, and delay-jitter. The notation $w(u, v)$ is used to denote the vector $(w_1(u, v), \ldots, w_k(u, v))$, which gives the k weights associated with the edge (u, v). Let p be a path in the network. We use $w_i(p)$ to denote the sum of the w_is of the edges on the path p.

$$w_i(p) = \sum_{(u,v) \in p} w_i(u, v)$$

By definition, $w(p) = (w_1(p), \ldots, w_k(p))$.

In the *multiconstrained path* (*k-MCP*) problem, we are to find a path p from a specified source vertex s to a specified destination vertex d such that

$$w_i(p) \leq c_i, \quad 1 \leq i \leq k \tag{10.6}$$

The c_is are specified QoS (quality of service) constraints. Note that Eq. (10.6) is equivalent to $w(p) \leq c$, where $c = (c_1, \ldots, c_k)$. A *feasible path* is any path that satisfies Eq. (10.6).

The *restricted shortest path* (*k-RSP*) problem is a related optimization problem in which we are to find a path p from s to d that minimizes $w_1(p)$ subject to

$$w_i(p) \leq c_i, \quad 2 \leq i \leq k$$

An algorithm is an ϵ-*approximation algorithm* (or simply, an *approximation algorithm*) for *k-MCP* iff the algorithm generates a source to destination path p that satisfies Eq. (10.6) whenever the network has a source to destination path p' that satisfies

$$w_i(p) \leq \epsilon * c_i, \quad 1 \leq i \leq k \tag{10.7}$$

where ϵ is a constant between 0 and 1.

Both the *k-MCP* and *k-RSP* problems for $k > 1$ are known to be NP-hard [6] and several pseudopolynomial-time algorithms, heuristics, and approximation algorithms have been proposed [7–9.] Jaffe [10] has proposed a polynomial-time approximation algorithm for 2-*MCP*. This algorithm, which uses a shortest path algorithm such as Dijkstra's [11], replaces the two weights on each edge by a linear combination of these two weights. The algorithm is expected to perform well when the two weights are positively correlated. Chen and Nahrstedt [12] use rounding to arrive at a polynomial-time approximation algorithm for *k-MCP*. Korkmaz and Krunz [13] propose a randomized heuristic that employs two phases. In the first phase a shortest path from each vertex of V to the destination vertex d is computed for each of the k weights as well as for a linear combination of all k weights. The second phase performs a randomized breadth-first search for a solution to the *k-MCP* problem. Yuan [14] has proposed two heuristics for *k-MCP*—limited granularity and limited path. By properly selecting the parameters for the limited granularity heuristic (LGH), this heuristic becomes an ϵ-approximation algorithm for *k-MCP*.

The papers [15–19] use rounding (up, down, and random) and interval partitioning to arrive at fully polynomial-time approximation schemes for *k-RSP*. Song and Sahni [20] use rounding (up), interval partitioning, and separation to develop fully polynomial-time approximation schemes for *k-MCP*. We focus on the work of Ref. [20] and this section is derived from Ref. [20].

10.6.2 Extended Bellman–Ford Algorithm

This is an extension of the well-known dynamic programming algorithm due to Bellman and Ford that is used to find shortest paths in weighted graphs [11]. The original Bellman–Ford algorithm was proposed for graphs in which each edge has a single weight. The extension allows for multiple weights (e.g., cost, delay, and delay-jitter).

Let u and v be two vertices in an instance of *k-MCP*. Let p and q be two different u to v paths. Path p is *dominated* by path q iff $w(q) \leq w(p)$ (i.e., $w_i(q) \leq w_i(p)$, $1 \leq i \leq k$).

In its pure form, the Bellman–Ford algorithm works in $n - 1$ (n is the number of vertices in the graph) rounds numbered 1 through $n - 1$. In round 1, the algorithm implicitly enumerates one-edge paths from the source vertex; then, in round 2, those with two edges are enumerated; and so on until finally paths with $n - 1$ edges are enumerated. Since no simple path has more than $n - 1$ edges, by the end of round $n - 1$, all simple paths have been (implicitly) enumerated. The enumeration of paths that have $i + 1$ edges is accomplished by considering all one-edge extensions of the enumerated i-edge paths. During the implicit enumeration, suboptimal paths (i.e., paths that are dominated by others) are eliminated. Suppose we have two paths p and q to vertex u and that p is dominated by q. If path p can be extended to a path that satisfies Eq. (10.6), then so also can q. Hence there is no need to retain p for further enumeration by path extension. Actual implementations rarely follow the pure Bellman–Ford paradigm and enumerate some paths of length more than $i + 1$ in round i.

Figure 10.4 gives the version of the Extended Bellman–Ford algorithm employed by Ref. [20]. This version is very similar to the version used by Yuan and others [14,21]. *PATH*(u) is a set of paths from the source s to vertex u. *PATH*(u) never contains two paths p and q for which $w(p) \leq w(q)$. Lines 12–14 initialize *PATH*(u) for all vertices u. The **for** loop of lines 16–20 attempts to implement the pure form of the Extended Bellman–Ford algorithm and performs the required $n - 1$ rounds (there is a provision to terminate in fewer rounds in case the previous round added a path to no *PATH*(u)). The method $Relax(u, v)$ extends the new[2] paths in *PATH*(u) by appending the edge (u, v). Feasible extended paths (i.e., those that satisfy the k constraints of Eq. [10.6]) are examined further. If v is the destination, the algorithm terminates as we have found a feasible source to destination path. Let the extended path $p||(u, v)$ be r. The inner **for** loop (lines 4–8) removes from *PATH*(v) all paths that are dominated by r (lines 7 and 8). This loop also verifies that r is not dominated by a path in *PATH*(v) (lines 5 and 6). Notice that if r is dominated by or equal to a path in *PATH*(v), r cannot dominate a path in *PATH*(v). Finally, in lines 9 and 10, r is added to *PATH*(v) only if it is not dominated by or equal to any path in *PATH*(v).

[2] A path is new iff it has not been the subject of a similar extension attempt on a previous round.

```
Relax($u$, $v$)
1.  for each new $p \in PATH(u)$ such that $w(p) + w(u, v) \leq c$ do
2.      if ($v = d$) return TRUE;
3.      Flag = TRUE;
4.      for each $q \in PATH(v)$ do
5.          if ($w(q) \leq w(p) + w(u, v)$)
6.              Flag = FALSE; Break; // exit inner for loop
7.          if ($(w(p) + w(u, v)) \leq w(q)$)
8.              remove $q$ from $PATH(v)$;
9.      if ($Flag == TRUE$)
10.         insert $p||(u, v)$ into $PATH(v)$; Change = TRUE;
11. return FALSE;

Extended Bellman–Ford($G$, $c$, $s$, $d$)
12. for $i = 0$ to $n - 1$ do
13.     $PATH(i) = NULL$;
14. $PATH(s) = \{s\}$;
15. Result = FALSE;
16. for $round = 1$ to $n - 1$ do
17.     Change = FALSE;
18.     for each edge $(u, v) \in E$ do
19.         if (Relax(u, v)) return "YES";
20.     if (Change == FALSE) return "NO";
21. return "NO";
```

FIGURE 10.4 Extended Bellman–Ford algorithm for *k-MCP*.

To see that the algorithm of Figure 10.4 is not a faithful implementation of the pure form of the Bellman–Ford algorithm, consider any iteration of the **for** loop of lines 16–20 (i.e., consider one round) and suppose that edge (u, v) is considered before edge (v, w) in the **for** loop of lines 13–14. Following the consideration of (u, v), *PATH*(v) possibly contains paths with *round* edges. So, when (v, w) is considered, *Relax* extends the paths in *PATH*(v) by a single edge (v, w) thereby permitting a path of length *round* $+ 1$ to be included in *PATH*(w). This lack of faithfulness in implementation of the pure Bellman–Ford algorithm does not affect the correctness of the algorithm and, in fact, agrees with the traditional implementation of the Bellman–Ford algorithm for the case when each edge has a single weight (i.e., $k = 1$) [11].

Another implementation point worth mentioning is that although we have defined *PATH*(u) to be a set of paths from the source to vertex u, it is more efficient to implement *PATH*(u) to be the set of weights (or more accurately, weight vectors $w(\,)$) of these paths. This, in fact, is how the algorithm is implemented in Ref. [14].

10.6.3 Rounding

Let $\delta_i = c_i * (1 - \epsilon)/n$, $2 \leq i \leq k$. Suppose we replace each $w_i(u, v)$ with the weight

$$w'_i(u, v) = \lceil \frac{w_i(u, v)}{\delta_i} \rceil * \delta_i$$

Let p be a path such that it satisfies Eq. (10.7). Then,

$$w'_i(p) < w_i(p) + n\delta_i \leq \epsilon c_i + (1 - \epsilon)c_i = c_i$$

So, algorithm *Extended Bellman–Ford* of Figure 10.4 when run with the edge weights $w_i(u, v)$ replaced by the weights $w'_i(u, v)$, $2 \leq i \leq k$ will find a feasible path (either p or some other feasible path). In an implementation of the rounding method, we actually replace each $w_i(u, v)$, $2 \leq i \leq k$ by

$$w_i''(u, v) = \lceil \frac{w_i(u, v)}{\delta_i} \rceil$$

and each c_i by $\lfloor c_i/\delta_i \rfloor$, $2 \leq i \leq k$. From the computation stand point, using the w_i's is equivalent to using the w_i''s.

Let $S = (n/(1 - \epsilon))^{k-1}$. In the w_i''s formulation, it is easy to see that $|PATH(u)| \leq S$. Hence the complexity of *Extended Bellman–Ford* when the w_i'' (equivalently, w_i') weights are used is $O(neS^2)$ and we have a fully polynomial-time approximation scheme for *k-MCP*. For the case $k = 2$, the complexity is $O(neS)$ if we employ the merge strategy of Horowitz and Sahni [4] to implement *Relax* (i.e., maintain *PATH*(u) in ascending order of w_1; extend the new paths in one step; then merge these extensions with *PATH*(v) in another step).

10.6.4 Interval Partitioning and Separation

In interval partitioning, we partition the space of $[w_2(p), w_3(p), \ldots, w_k(p)]$ values into buckets of size $[\delta_2, \delta_3, \ldots, \delta_k]$. *PATH*$(u)$ is maintained so as to have at most one path in each bucket. When a *Relax* step attempts to put a second path into a bucket, only the path with the smaller w_1 value is retained. When the δ_is are chosen as in Section 10.6.3, we get a fully polynomial-time approximation scheme. By choosing larger values for the δ_is, we lose the guarantee of an ϵ-approximate solution but we reduce the run time. We use the term *interval partitioning heuristic* (IPH) to refer to the interval partitioning algorithm in which the δ_is are chosen arbitrarily.

Figure 10.5 gives the relax method used by IPH. The driver *Extended Bellman–Ford* is unchanged. By choosing the number of buckets (equivalently, the bucket size) as in Section 10.6.3, we get a fully polynomial-time ϵ-approximation scheme. The proof of this claim is quite similar to that of the proof provided in Section 10.6.3.

Theorem 10.1

IPH is an ϵ-approximation algorithm for k-MCP when the bucket size is chosen as in Section 10.6.3.

In the separation method, *PATH*(u) is such that no two paths of *PATH*(v) are within $\delta_i/2$ of their w_i values for $2 \leq i \leq k$. So, if we attempt to add to *PATH*(v) a path q such that $w_i(p)+\delta_i/2 \leq w_i(q) \leq w_i(p)+\delta_i/2$, $2 \leq i \leq k$, where $p \in PATH(v)$, then only the path with the smaller w_1 value is retained.

Since separation comes with greater implementation overheads than associated with interval partitioning [20] focuses on the interval partitioning method for k-MCP.

```
RelaxIPH($u$, $v$)
for each new $p \in PATH(u)$ such that $w(p) + w(u, v) \leq c$ do
    if ($v = d$) return TRUE;
    Let $r = p||(u, v)$;
    Let $q \in PATH\,(v)$ such that $r$ and $q$ fall in the same bucket;
    if (there is no such $q$)
        Add $r$ to $PATH\,(v)$; Change = TRUE;
    else if ($w_1(r) < w_1(q)$)
        Replace $q$ by $r$ in $PATH(v)$; Change = TRUE;
return FALSE;
```

FIGURE 10.5 Relax method for IPH.

10.6.5 The Heuristics of Yuan [14]

The LGH of Yuan [14] combines the interval partitioning and rounding methods. *PATH*(v) is represented as a $(k-1)$-dimensional array with each array position representing a bucket of size $[s_2, s_3, \ldots, s_k]$. As in the pure form of interval partitioning, each bucket can have at most one path. However, unlike interval partitioning, the exact w_i values of the retained path are not stored. Instead, the w_i values, $2 \leq i \leq k$ are rounded up to the maximum possible for the bucket; the smallest w_1 value of the paths that fall into a bucket is stored in the bucket. Note that because of the rounding of the w_i values, $2 \leq i \leq k$, we do not store these values along with the path; they may be computed as needed from the bucket indexes.

We may regard the LGH as one with delayed rounding; the rounding done at the outset when the traditional rounding method is used, is delayed to the time a path is actually constructed. By incorporating buckets, we eliminate the need to store the w_i, $2 \leq i \leq k$, values stored explictly with each path when either the rounding or interval partitioning methods are used. Although there is a reduction in space (by a factor of k) on a per path basis, the array of buckets used to implement each *PATH*(u) needs $\prod_{2\leq i\leq k} c_i/s_i$ space, whereas when the w_is are explicitly stored, the space requirements can be reduced to $O(k *$ total number of paths stored). The time complexity of LGH is $O(ne\prod_{2\leq i\leq k} c_i/s_i)$.

Note that when $s_i = \delta_i$, $2 \leq i \leq k$, the LGH becomes an ϵ-approximation algorithm.

The limited path heuristic (LPH) of Yuan [14] limits the size of *PATH*(v) to be X, where X is a specified parameter. It differs from *Extended Bellman–Ford* (Figure 10.4) only in that line 9 is changed to **if** (*Flag* == *True* && |*PATH*$(v)| < X$). With this modification, the complexity of *Extended Bellman–Ford* becomes $O(neX^2)$. The success of LPH hinges on the expectation that the first X nondominated paths, to vertex v, found by *Extended Bellman–Ford* are more likely to lead to a feasible path to the destination than subsequent paths to v. In a pure implementation of the Bellman–Ford method (which Figure 10.4 is not), this expectation may be justified with the expectation that paths to nondestination vertices with a smaller number of edges (these are found first in a pure Bellman–Ford algorithm) are more likely to lead to a feasible path to the destination than those with a larger number of edges.

10.6.6 Generalized Limited Path Heuristic

LPH limits the number of paths in *PATH*(u) to be at most X. In generalized limited path heuristic (GLPH), the constraint on the number of paths is

$$\sum_{u\in V, u\neq s} |PATH(u)| \leq (n-1) * X$$

While both LPH and GLPH place the same limit on the total number of paths retained (i.e., $(n-1)*X$), LPH accomplishes this by explicitly restricting the number of paths in each *PATH*(u), $u \neq s$ to be no more than X.

To ensure a performance at least as good as that of LPH, GLPH ensures that each *PATH*(u) maintains a superset of the *PATH*(u) maintained by LPH. So, GLPH permits the size of a *PATH*(u) to exceed X so long as the sum of the sizes is no more than $(n-1) * X$. When the sum of the sizes equals $(n-1) * X$, we continue to add paths to those *PATH*(u)s that have fewer than X paths. However, each such addition is accompanied by the removal of a path that would not be in any *PATH*(v) of LPH.

10.6.7 Hybrid Interval Partitioning Heuristics (HIPHs)

Although IPH becomes an ϵ-approximation algorithm when the bucket size is chosen appropriately, LPH is expected to perform well on many real-world networks because we expect paths with a small number of edges to be more likely to lead to feasible source–destination paths than those with a large number of edges. In this section we describe four hybrid heuristics: HIPH1, HIPH2, HIPH3, and HIPH4.

HIPH1 and HIPH2 combine IPH and LPH into a unified heuristic that has the merits of both. HIPH1 maintains two sets of paths for each vertex $u \in V$. The first set *PATH*(u) is limited to have at most X

```
RelaxHIPH1$(u, v)$
for each new $p \in PATH(u)$ such that $w(p) + w(u, v) \leq c$ do
    if $(v = d)$ return TRUE;
    Flag = TRUE;
    for each $q \in PATH(v)$ do
        if $(w(p) + w(u, v) \geq w(q))$
            Flag = FALSE; Break; // exit for loop
        if $((w(p) + w(u, v)) < w(q))$
            remove $q$ from $PATH(v)$;
    if $(Flag == TRUE)$
        if $(|PATH(v)| < X)$
            insert $p||(u, v)$ into $PATH(v)$; Change = TRUE;
        else
            do lines 3-8 of RelaxIPH using ipPATH in place of PATH;
// Relax using ipPATH in place of PATH
return RelaxIPH$(u, v)$;
```

FIGURE 10.6 Relax method for HIPH1.

paths. This set is a faithful replica of *PATH*(u) as maintained by LPH. The second set, *ipPATH*(u), uses interval partitioning to store additional paths found to vertex u. For the source vertex s, *PATH*$(s) = \{s\}$ and *ipPATH*(s)= $\emptyset$. Figure 10.6 gives the new relax method employed by HIPH1. It is easy to see that if on entry to *RelaxHIPH1*, *PATH*(u) as maintained by HIPH1 is the same as that maintained by the relax method of LPH, then on exit, *PATH*(v) is the same for both HIPH1 and LPH. Since both heuristics start with the same *PATH*(u) for all u, both maintain the same *PATH*(u) sets throughout. Hence HIPH1 produces a feasible solution whenever LPH does. Furthermore, because HIPH1 maintains additional paths in *ipPATH*( ), it has the potential to find feasible source-to-destination paths even when LPH fails to do so. It is easy also to see that when bucket size is selected as in Section 10.6.3, HIPH1 is an ϵ-approximation algorithm.

Theorem 10.2

HIPH1 is an ϵ-approximation algorithm for k-MCP when the bucket size for ipPATH( ) is chosen as in Section 10.6.3. Further, for any given X, HIPH1 finds a feasible source-to-destination path whenever LPH finds such a path.

HIPH2 is quite similar to HIPH1. In HIPH1 the extension $r = p||(u, v)$ of a path $p \in$ *ipPATH*(u) can be stored only in *ipPATH*(v). In HIPH2, however, this extension is stored in *PATH*(v) whenever $|PATH(v)| < X$. When $|PATH(v)| = X$, lines 4–8 of *RelaxIPH* are applied (using *ipPATH*(v) in place of *PATH*(v)) to determine the fate of r. With this change, *PATH*(u) as maintained by LPH may not be the same as that maintained by HIPH2. However, by choosing the bucket size for *ipPATH*(u) as in Section 10.6.3, HIPH2 becomes an ϵ-approximation algorithm.

Theorem 10.3

HIPH2 is an ϵ-approximation algorithm for k-MCP when the bucket size for ipPATH( ) is chosen as in Section 10.6.3.

HIPH3 and HIPH4 are the GLPH analogs of HIPH1 and HIPH2; that is they are based on GLPH rather than LPH.

Theorem 10.4

HIPH3 and HIPH4 are ϵ-approximation algorithms for k-MCP when the bucket size for ipPATH( ) is chosen as in Section 10.6.3.

Dataset	*Algorithm*							
	LGH	LPH	IPH	GLPH	HIPH1	HIPH2	HIPH3	HIPH4
8 × 8 mesh, $k = 2$, unbiased	-	8*	-	4	8	8	4	4
16 × 16 mesh, $k = 2$, unbiased	-	-	-	8*/16	16*	16*	8*/16	8*/16
8 × 8 mesh, $k = 2$, biased	-	-	-	8	8*	4*/8	2*/4	2*/4
16 × 16 mesh, $k = 2$, biased	-	-	1	16	16*	1	8	1
Power-law, $k = 2$, unbiased	-	4*/8	16*	2	4*/8	4	1*/2	1*/2
Power-law, $k = 2$, biased	-	4*/8	16*	2	4*/8	4	2	2
ADC, $k = 2$	-	16	1*	8	16	1*/8	8	1*/8

FIGURE 10.7 Smallest X at which competitive ratio becomes 1.0 [20].

10.6.8 Performance Evaluation

The *existence ratio* (ER) and *competitive ratio* (CR) are defined, respectively, by Yuan [14] to be the number of routing requests satisfied by the extended Bellman–Ford algorithm divided by the total number of routing requests and the number of routing requests satisfied by a heuristic divided by the number satisfied by the extended Bellman–Ford algorithm. For example, if we make 500 routing requests, 100 of which are satisfiable, the ER is $100/500 = 0.2$. If LPH is able to find a feasible path for 80 of the 100 requests for which such a path exists, the CR of LPH is $80/100 = 0.8$.

Song and Sahni [20] report on an extensive simulation study involving mesh [14], power-law [22], and augmented directed chain (ADC) [20] networks. Figure 10.7 gives the smallest of the tested X values for which the CR becomes 1.0. For the case when $k = 2$, X is the bound placed on $|PATH(u)|$ and $|ipPATH(u)|$. In particular, for LGH, X is the number of positions in the one-dimensional array used to represent each *PATH(u)* and for IPH, X is the number of intervals for each *PATH(u)*. GLPH working on a network with n vertices is able to store at most $X*(n-1)$ paths, which is the maximum number of paths in all *PATH(u)* lists of LPH. For the hybrid heuristics HIPH1 and HIPH2, $|PATH(u)| \leq X$ and $|ipPATH(u)| \leq X$. For HIPH3 and HIPH4, $\sum |PATH(u)| \leq X*(n-1)$ and $|ipPATH(u)| \leq X$. Note that since every heuristic other than LGH stores both w_1 and w_2 for each path while LGH stores only w_1, the worst-case space requirements of LGH for any X are one-half for LPH and GLPH and one-fourth for HIPH1 through HIPH4. In Figure 10.7, X values labeled with an "$*$" indicate that the CR becomes almost 1.0, more precisely, larger than 0.99. So, for example, the entry $8*/16$ for GLPH, HIPH3, and HIPH4 working on 16 × 16 unbiased meshes means that these heuristics achieved a CR very close to 1.0 when $X = 8$ and a CR of 1.0 when $X = 16$. The "$-$" in the entry for 16 × 16 unbiased meshes for LGH means that the CR ratio for LGH did not become close to 1.0 for any of the tested X values.

10.6.9 Summary

All of the studied *k-MCP* heuristics, with the exception of GLPH, become ϵ-approximation schemes when the bucket size is chosen as in Section 10.6.3. Although GLPH has the same bound on total memory as does the limited path heuristic LPH of Ref. [14], GLPH provides better CR; in fact, GLPH finds a feasible path whenever LPH does and is able to find feasible solutions for several instances on which LPH fails to do so. The IPH heuristic achieves significantly better CRs than are achieved by the LGH of Ref. [14]. LPH and GLPH do well on graphs in which there is at least one feasible path that has a small number of edges. On ADCs that do not have such feasible paths, LPH and GLPH provide miserable performance [20]. The hybrid heuristics HIPH1 through HIPH4 combine the merits of IPH (ϵ-approximation when bucket size is chosen properly) and LPH and GLPH (guaranteed success when the graph has a feasible path with few edges). Of the four hybrid heuristics, HIPH4 performed best in the experiments of Ref. [20].

References

[1] Sahni, S., General techniques for combinatorial approximation, *Oper. Res.*, 25(6), 920, 1977.

[2] Raghavan, R. and Thompson, C., Randomized rounding: a technique for provably good algorithms and algorithmic proof, *Combinatorica*, 7, 365, 1987.

[3] Horowitz, E., Sahni, S., and Rajasekeran, S., *Fundamentals of Computer Algorithms,* W. H. Freeman, New York, 1998.

[4] Horowitz, E. and Sahni, S., Computing partitions with applications to the knapsack problem, *JACM*, 21(2), 277, 1974.

[5] Ibarra, O. and Kim, C., Fast approximation algorithms for the knapsack and sum of subset problems, *JACM*, 22(4), 463, 1975.

[6] Garey, M. and Johnson, D., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[7] Chen, S. and Nahrstedt, K., An overview of quality-of-service routing for the next generation high-speed networks: problems and solutions, *IEEE Network Mag.*, 12, 64, 1998.

[8] Kuipers, F., Van Mieghem, P., Korkmaz, T., and Krunz, M., An overview of constraint-based path selection algorithms for QoS routing, *IEEE Commun. Mag.*, 40(12), 2002, Pages 50–55.

[9] Younis, O. and Fahmy, S., Constraint-based routing in the Internet: Basic principles and recent research, *IEEE Commun. Surv. Tutorials*, 5(1), 2, 2003.

[10] Jaffe, J. M., Algorithms for finding paths with multiple constraints, *Networks*, 14, 95, 1984.

[11] Sahni, S., *Data Structures, Algorithms, and Applications in C++*, 2nd ed., Silicon Press, Summit, New Jersy, 2005.

[12] Chen, S. and Nahrstedt, K., On finding multi-constrained paths, *IEEE Int. Conf. Communications (ICC'98)*, June 1998.

[13] Korkmaz, T. and Krunz, M., A randomized algorithm for finding a path subject to multiple QoS requirements, *Comput. Networks*, 36, 251, 2001.

[14] Yuan, X., Heuristic algorithms for multiconstrained quality of service routing, *IEEE/ACM Trans. Networking*, 10(2), 244, 2002.

[15] Chen, S., Song, M., and Sahni, S., Two techniques for fast computation of constrained shortest paths, *Proc. IEEE GLOBECOM 2004*, 2004.

[16] Goel, A., Ramakrishnan, K. G., Kataria, D., and Logothetis, D., Efficient computation of delay-sensitive routes from one source to all destinations, *IEEE INFOCOM'01*, 2001.

[17] Hassin, R., Approximation schemes for the restricted shortest path problem, *Math. Oper. Res.*, 17(1), 36, 1992.

[18] Korkmaz, T. and Krunz, M., Multi-constrained optimal path selection, *IEEE INFOCOM'01*, April 2001.

[19] Lorenz, D. H. and Raz, D., A simple efficient approximation scheme for the restricted shortest path problem, *Oper. Res. Lett.*, 28, 213, 2001.

[20] Song, M. and Sahni, S., Approximation algorithms for multiconstrained quality-of-service routing, IEEE Transactions on Computers, 55(5), 603–617, 2006.

[21] Widyono, R., The design and evaluation of routing algorithms for real-time channels, TR-94-024, Internaltional Computer Science Institute, UC Berkeley, 1994.

[22] Faloutsos, M., Faloutsos, P., and Faloutsos, C., On power-law relationships of the Internet topology, *ACM Proc. SIGCOMM '99*, 1999.

11

Asymptotic Polynomial-Time Approximation Schemes

Rajeev Motwani
Stanford University

Liadan O′Callaghan
Google

An Zhu
Google

11.1 Introduction

We illustrate the concept of asymptotic (fully) polynomial-time approximation schemes (APTAS, AFPTAS) via a study of the bin packing problem. We discuss in detail an APTAS due to Fernandez de la Vega and Lueker [1] and an AFPTAS due to Karmakar and Karp [2]. Many of the algorithmic and analytical techniques described in this chapter can be applied elsewhere in the development and study of other polynomial-time approximation schemes. We conclude with a brief survey of other bin packing-related results and other examples of APTAS and AFPTAS.

We first introduce the classic bin packing problem, which is $\mathcal{NP}$-complete. Informally, we are given a collection of items of sizes between 0 and 1. We are required to pack them into bins of unit size so as to minimize the number of bins used. Thus, we have the following minimization problem.

BIN PACKING **(BP):**

- [**Instances**] $I = \{s_1, s_2, \ldots, s_n\}$, such that $\forall i,\ s_i \in [0, 1]$.
- [**Solutions**] A collection of subsets $\sigma = \{B_1, B_2, \ldots, B_k\}$ which is a disjoint partition of I, such that $\forall i,\ B_i \subset I$ and $\sum_{j \in B_i} s_j \leq 1$.
- [**Value**] The value of a solution is the number of bins used, or $f(\sigma) = |\sigma| = k$.

BIN PACKING is a perfect illustration of why sometimes the absolute performance ratio is not the best possible definition of the performance guarantee for an approximation algorithm. Recall that the absolute performance ratio, a.k.a. the approximation ratio, of an algorithm A for a minimization problem is defined as

$$R_A = \inf\left\{ r \mid R_A(I) = \frac{A(I)}{OPT(I)} < r, \forall I \right\}$$

where $A(I)$ and $OPT(I)$ denote the value of algorithm A's solution and the optimal solution for instance I, respectively.[1] Note that the problem of deciding if an instance of BIN PACKING has a solution with two bins is $\mathcal{NP}$-complete—this is exactly the PARTITION problem [3]. This implies that no algorithm can guarantee an approximation ratio better than 3/2 for BIN PACKING. Consequently, no approximation schemes, PTAS or FPTAS [4], exist for BIN PACKING.

The hardness of 3/2 comes from the fact that we cannot decide between two or three bins, a difference of one bin only. It is the small value of the optimum solution that makes the approximation ratio appear to be large; the approximation ratio is misleading, since on larger instances the ratio could still be bounded by a small constant. Therefore, we introduce the asymptotic performance ratio:

Definition 11.1

The asymptotic performance ratio, R_A^∞, *of an approximation algorithm A for an optimization problem is*

$$R_A^\infty = \inf\{r \mid \exists N_0,\ R_A(I) \le r \ \text{ for all } I \text{ with } OPT(I) \ge N_0\}$$

For BIN PACKING, the 3/2-hardness result does not preclude the existence of *asymptotic* approximation schemes, which give an approximation factor that approaches 1 in the limit:

Definition 11.2

An Asymptotic PTAS (APTAS) *is a family of algorithms* $\{A_\epsilon \mid \epsilon > 0\}$ *such that each* A_ϵ *runs in time polynomial in the length of the input and* $R_{A_\epsilon}^\infty \le 1 + \epsilon$.

Definition 11.3

An Asymptotic FPTAS (AFPTAS) *is a family of algorithms* $\{A_\epsilon \mid \epsilon > 0\}$ *such that each* A_ϵ *runs in time polynomial in the length of the input and* $1/\epsilon$, *while* $R_{A_\epsilon}^\infty \le 1 + \epsilon$.

In this chapter we present two algorithms, an APTAS and an AFPTAS, due to Fernandez de la Vega and Lueker [1] and Karmakar and Karp [2], respectively, for BIN PACKING. The algorithmic and analytic tools demonstrated here are widely applicable to the study and development of approximation schemes. Some of the techniques, such as interval partitioning, have been applied to similar problems such as Multiprocessor Scheduling, Knapsack [3] and various packing-related problems and their generalizations. Other techniques are more general and apply in a broader range of problem settings; for instance, linear programming is a very powerful tool and has been used with enormous success throughout operations research, management science, and theoretical computer science.

The rest of this chapter is organized as follows: Section 11.2 presents a summary of the techniques used in the two algorithms; Section 11.3 presents the APTAS, Section 11.4 presents the AFPTAS, and finally, Section 11.5 summarizes some other results related to BIN PACKING, and lists some other examples of APTAS and AFPTAS.

11.2 Summary of Algorithms and Techniques

The first result we present is due to Fernandez de la Vega and Lueker [1], who provided an APTAS for BIN PACKING that runs in *linear time* and has $A_\epsilon(I) \le (1+\epsilon) \cdot OPT(I) + 1$. To be more specific, the running time is linear in the size of the input instance I but is severely exponential in ϵ. Note that the reason this scheme is an APTAS, and not a PTAS, is the additive error term of 1 in the approximation bound. The basic techniques used in this result may be summarized as follows:

[1] $R_A(I)$, the absolute performance ratio of algorithm A on an input instance I, is defined as $OPT(I)/A(I)$ for maximization problems. Such a definition ensures that $R_A(I) \ge 1$ always.

- Separate handling of “small” items.
- Discretization via interval partitioning or linear grouping.
- Rounding of “fractional” solutions.

We then present the modification of this result due to Karmakar and Karp [2], which leads to an AFPTAS for BIN PACKING. They give an approximation scheme with a performance guarantee similar to the one described above, with running time improved to $O(\frac{n \log n}{\epsilon^8})$.

We now derive the results described above. Our presentation combines the methods of Fernandez de la Vega and Lueker with those of Karmakar and Karp, as the two techniques share many of the same basic tools. The general approach used in both techniques is as follows: We first define a restricted version of the problem in which all items are of at least some minimum size, and the item sizes can only take on a few distinct values. This new version of BIN PACKING turns out to be reasonably easy to solve. Then we provide a two-step reduction from the original problem instance to a restricted problem instance. The first step is to pull out the “small” items; it is shown that given any packing of the remaining items, the small items can be added back in without a significant increase in the number of bins used. The second step is to divide the item sizes into m intervals, and replace all items in the same interval by items of the same size. It turns out that this “linear grouping” affects the value of the optimal solution only marginally. In the next two sections, we consider each of these ingredients in turn and finally show how they can be combined to produce an APTAS and then an AFPTAS.

11.3 Asymptotic Polynomial-Time Approximation Scheme

Definition 11.4

For any instance $I = \{s_1, \ldots, s_n\}$, let $SIZE(I) = \sum_{i=1}^{n} s_i$ denote the total size of the n items.

Recall that $OPT(I)$ denotes the value of the optimal solution, i.e., the minimum number of unit size bins needed to pack the items. We now give two inequalities relating these quantities.

Lemma 11.1

$SIZE(I) \leq OPT(I) \leq |I| = n.$

Proof

In the optimal solution, at best each bin is filled to its maximum capacity, i.e., 1. Thus, the total number of bins needed is at least $SIZE(I)/1$, proving $SIZE(I) \leq OPT(I)$. Since each item is of size between 0 and 1, putting each item in a separate bin is clearly a feasible (if not optimal) solution, proving $OPT(I) \leq |I| = n$. □

Lemma 11.2

$OPT(I) \leq 2 \cdot SIZE(I) + 1.$

Proof

Prove by contradiction. Suppose this is not the case, i.e., there exists an instance I, where $OPT(I) > 2 \cdot SIZE(I) + 1$. Then in the optimal solution, there must exist at least two bins that are at least half empty. Otherwise, we have at least $OPT(I) - 1$ number of bins that are at least half full, i.e., $(OPT(I) - 1)/2 \leq SIZE(I)$, which contradicts our initial assumption. Now the fact that we have two bins at least half empty contradicts the assumption that we have an optimal solution. We could have easily combined the two bins into one, and reduce the number of bins used by 1. Thus, our initial assumption must be false, proving the lemma. □

We will represent an instance I as an *ordered* list of items $I = s_1 s_2 \ldots s_n$ such that $1 \geq s_1 \geq s_2 \geq \cdots \geq s_n > 0$.

Definition 11.5

Let $I_1 = x_1 x_2 \dots x_n$ and $I_2 = y_1 y_2 \dots y_n$ be two instances of equal cardinality. The instance I_1 is said to **dominate** *the instance I_2, or $I_1 \geq I_2$, if it is the case that $x_i \geq y_i$, for all i.*

The following lemma follows from the fact that any feasible packing of I_1 gives a feasible packing of I_2, using the same number of bins.

Lemma 11.3

Let I_1 and I_2 be two instances of equal cardinality such that $I_1 \geq I_2$. Then, $SIZE(I_1) \geq SIZE(I_2)$ and $OPT(I_1) \geq OPT(I_2)$.

We define a restricted version of BIN PACKING as follows. Suppose that the item sizes in I take on only m distinct values. Now the instance I can be represented as a multiset of items which are drawn from these m types of items.

Definition 11.6

Suppose that we are given m distinct item sizes $V = \{v_1, \dots, v_m\}$, such that $1 \geq v_1 > v_2 > \cdots > v_m > 0$, and an instance I of items whose sizes are drawn only from V. Then, we can represent I as multiset $M_I = \{n_1 : v_1, n_2 : v_2, \dots, n_m : v_m\}$, where n_i is a nonnegative integer denoting the number of items in I of size v_i.

It follows that $|M_I| = \sum_{i=1}^{m} n_i = n$, $SIZE(M_I) = \sum_{i=1}^{m} n_i v_i = SIZE(I)$ and $OPT(M_I) = OPT(I)$. We now define RBP, the restricted version of BIN PACKING.

Definition 11.7

For all $0 < \delta < 1$ and positive integers m, the problem $RBP[\delta, m]$ is defined as BIN PACKING *restricted to instances where the item sizes take on at most m distinct values and the size of each item is at least δ.*

Next we show how to approximately solve *RBP* via a linear programming formulation.

11.3.1 Restricted Bin Packing

Assume that δ and m are fixed independently of the input size n. The input instance for $RBP[\delta, m]$ is a multiset $M = \{n_1 : v_1, n_2 : v_2, \dots, n_m : v_m\}$, such that $1 \geq v_1 > v_2 > \cdots > v_m \geq \delta$. Let $n = |M| = \sum_{i=1}^{n} n_i$. In the following discussion, we will assume that the underlying set V for M is fixed. Note that, given M, it is trivial to determine V and verify that M is a valid instance of $RBP[\delta, m]$.

Consider a packing of some subset of the items in M into a unit size bin. We can denote this packing by a multiset $\{b_1 : v_1, b_2 : v_2, \dots, b_m : v_m\}$, such that b_i is the number of items of size v_i that are packed into the bin. More concisely, having fixed V, we can denote the packing by the m-vector $B = (b_1, \dots, b_m)$ of nonnegative integers. We will say that two bins packed with items from M are of the same *type* if the corresponding packing vectors are identical:

Definition 11.8

A bin type *T is an m-vector $(T_1, \dots, T_m)$ of nonnegative integers such that $\sum_{i=1}^{m} T_i v_i \leq 1$.*

Having fixed the set V, the collection of possible bin types is fully determined and is finite, because each T_i in T must take on an integer value from 0 to $\lfloor 1/v_i \rfloor$. Let $T^1, \dots, T^q$ denote the set of all legal bin types with respect to V. Here q, the number of distinct types, is a function of δ and m. We bound the value of q in the following lemma:

Lemma 11.4

Let $k = \lfloor \frac{1}{\delta} \rfloor$. Then

$$q(\delta, m) \leq \binom{m+k}{k}$$

Proof

Each type vector $T^t = (T_1^t, \ldots, T_m^t)$ has the property that, for all i, $T_i^t \geq 0$ and $\sum_{i=1}^{m} T_i^t v_i \leq 1$. It follows that $\sum_{i=1}^{m} T_i^t \leq k$, since we have a lower bound of δ on the values v_i in V. Thus, each type vector corresponds to a way of choosing m nonnegative integers whose sum is at most k. This is the same as choosing $m + 1$ nonnegative integers whose sum is exactly k. The number of such choices is an upper bound on the value of q. A standard counting argument now gives the desired bound. □

Consider an arbitrary feasible solution x to an instance M of $RBP[\delta, m]$. Each packed bin in this solution can be classified as belonging to one of the $q(\delta, m)$ possible types of packed bins. The solution x can therefore be specified completely by a vector giving the number of bins of each of the q types.

Definition 11.9

A feasible solution x to an instance M of $RBP[\delta, m]$ is a q-vector of nonnegative integers, say $x = (x_1, \ldots, x_q)$, where x_t denotes the number of bins of type T^t used in x.

Note that not all q-vectors correspond to a feasible solution. A feasible solution must guarantee, for each i, that exactly n_i items of size v_i are packed in the various copies of the bin types. The feasibility condition can be phrased as a series of linear equations as follows:

$$\forall i \in \{1, \ldots, m\}, \quad \sum_{t=1}^{q} x_t T_i^t = n_i$$

Let the matrix A be a $q \times m$ matrix whose tth row is the type vector T^t, and $\vec{n} = (n_1, \ldots, n_m)$ denote the multiplicities of the various item sizes in the input instance M. Then the above set of equations can be concisely expressed as $\vec{x}.A = \vec{n}$. The number of bins used in the solution x is simply $\vec{x} \cdot \vec{1} = \sum_{t=1}^{q} x_t$, where $\vec{1}$ denotes all-ones vector. In fact, we have proved the following lemma.

Lemma 11.5

The optimal solution to an instance M of $RBP[\delta, m]$ is exactly the solution to the following integer linear program ILP(M)

$$\begin{aligned} \text{minimize} \quad & \vec{x} \cdot \vec{1} \\ \text{subject to} \quad & \\ & \vec{x} \geq 0 \\ & \vec{x} \cdot A \geq \vec{n} \end{aligned}$$

We have replaced the equations by inequalities, but, since a packing of a superset of M can always be converted into a packing of M using the same number of bins, the validity of the lemma is unaffected. It is also worth noting that the matrix A is determined completely by the underlying set V; the vector $\vec{n}$, however, is not determined *a priori* but depends on the instance M.

How easy is it to obtain this integer program? Note that the number of constraints in ILP(M) is exponentially large in terms of δ and m. However, we are going to assume that both δ and m are constants which are fixed independently of the length of the input, which is n. Thus, ILP(M) can be obtained in time linear in n, given any instance M of cardinality n.

How about solving ILP? Recall that the integer programming problem is $\mathcal{NP}$-complete in general [3]. However, there is an algorithm due to Lenstra [5–7] that solves any integer linear program in time linear in the number of constraints, provided the number of variables is fixed. This is exactly the situation in ILP: the number of variables q is fixed independent of n, as is the number of constraints, which is $q + m$. Thus, we can solve ILP exactly in time independent of n. (A more efficient algorithm for approximately solving ILP will be described in a later section.) The following theorem results. Here $f(\delta, m)$ is some constant which depends only on δ and m.

Theorem 11.1

Any instance of RBP$[\delta, m]$ can be solved in time $O(n + f(\delta, m))$.

11.3.2 Eliminating Small Items

We now present the second ingredient of the APTAS devised by Fernandez de la Vega and Lueker: the separate handling of small items. It is shown that if we have a packing of all items except those whose sizes are bounded from above by δ, then it is possible to add the small items back in without much increase in the number of bins. This fact is summarized in the following lemma; the rest of this subsection is devoted to the proof of this lemma.

Lemma 11.6

Fix some constant $\delta \in (0, \frac{1}{2}]$. Let I be an instance of BIN PACKING *and suppose that all items of size greater than δ have been packed into β bins. Then it is possible to find in linear time a packing for I which uses at most* $\max\{\beta, (1+2\delta)\cdot OPT(I)+1\}$ *bins.*

Proof

The basic idea is to start with the packing of the "large" items and to use the greedy algorithm First Fit to pack the "small" items into the empty space in the β bins.

First Fit (*FF*) is a classic bin packing algorithm of historical importance, as we shall see later. The algorithm is as follows. We are given the set of items in an arbitrary order, and we start with zero bins. For each item in the list, we consider the existing bins (if any) in order and place the item in the first bin that can accommodate it. If no existing bin can accommodate it, we make a new bin after all the existing ones, and put the item in the new bin.

To use First Fit to add the small items into an existing packing of the large ones, we can start by numbering the β bins in an arbitrary fashion, and also ordering the small items arbitrarily. Then we run First Fit as usual using this ordering to decide where each small item will be placed. If at some point the small items do not fit into any of the currently available bins, a new bin is initiated.

In the best case, the small items can all be greedily packed into the β bins which were open initially. Clearly, the lemma is valid in that case. Suppose now that some new bins were required for the small items. We claim that at the end of the entire process, each of the bins used for packing I has at most δ empty space in it, with the possible exception of at most one bin. To see why this claim holds, note that at the moment when the first new bin was started, each of the original bins must have had at most δ free space. Next, observe that whenever another new bin was opened, no earlier bin could have had more than δ free space. Therefore, at every moment, at most one bin had more than δ free space.

Let $\beta' > \beta$ be the total number of bins used by *FF*. We are guaranteed that all the bins, except one, are at least $1-\delta$ full. This implies that *SIZE*$(I) \geq (1-\delta)(\beta'-1)$. But we know that *SIZE*$(I) \leq OPT(I)$, implying that

$$\beta' \leq \frac{1}{1-\delta} OPT(I) + 1 \leq (1+2\delta)\cdot OPT(I) + 1$$

and we have the desired result. □

11.3.3 Linear Grouping

The final ingredient needed for the APTAS is called interval partitioning or linear grouping. This is a technique for converting an instance I of BIN PACKING into an instance M of *RBP*$[\delta, m]$, for an appropriate choice of δ and m, without changing the value of the optimal solution too much. Let us assume for now that all the items in I are of size at least δ, for some choice of $\delta \in (0, \frac{1}{2}]$. All that remains is to show how to obtain an instance where the item sizes take on only m different values. First, let us fix some parameter k, a nonnegative integer to be specified later. We now show how to convert an instance of *RBP*$[\delta, n]$ into an instance of *RBP*$[\delta, m]$, for $m = \lfloor n/k \rfloor$.

Definition 11.10

Given an instance I of RBP$[\delta, n]$ and a parameter k, let $m = \lfloor n/k \rfloor$. Define the groups of items $G_i = s_{(i-1)k+1} \ldots s_{ik}$, for $i = 1, \ldots, m$, and let $G_{m+1} = s_{mk+1} \ldots s_n$.

Here, the group G_1 contains the k largest items in I, G_2 the next k largest items, and so on. The following fact is an easy consequence of these definitions.

Fact 11.1

$G_1 \geq G_2 \geq \cdots \geq G_m$.

From each group G_i we can obtain a new group of items H_i by increasing the size of each item in G_i to that of the largest item in that group. The following fact is also obvious.

Definition 11.11

Let $v_i = s_{(i-1)k+1}$ be the largest item in group G_i. Then the group H_i is a group of $|G_i|$ items, each of size v_i. In other words, $H_i = v_i v_i \ldots v_i$ and $|H_i| = |G_i|$.

Fact 11.2

$H_1 \geq G_1 \geq H_2 \geq G_2 \geq \cdots \geq H_m \geq G_m$ *and* $H_{m+1} \geq G_{m+1}$.

The entire point of these definitions is to obtain two instances of *RBP*$[\delta, m]$ such that their optimal solutions bracket the optimal solution for I. These instances are defined as follows.

Definition 11.12

Let the instance $I_{LO} = H_2 H_3 \ldots H_{m+1}$ and $I_{HI} = H_1 H_2 H_3 \ldots H_{m+1}$.

Note that I_{LO} is an instance of *RBP*$[\delta, m]$. Moreover, it is easy to see that $I \leq I_{HI}$. We now present some properties of these three instances.

Lemma 11.7

$$OPT(I_{LO}) \leq OPT(I) \leq OPT(I_{HI}) \leq OPT(I_{LO}) + k$$
$$SIZE(I_{LO}) \leq SIZE(I) \leq SIZE(I_{HI}) \leq SIZE(I_{LO}) + k$$

Proof

First, observe that

$$I_{LO} = H_2 H_3 \ldots H_m H_{m+1} \leq G_1 G_2 \ldots G_{m-1} X$$

where X is any set of $|H_{m+1}|$ items from G_m. The right-hand side of this inequality is a subset of I, and so, from Lemma 11.3, $OPT(I_{LO}) \leq OPT(I)$ and $SIZE(I_{LO}) \leq SIZE(I)$. Similarly, since $I \leq I_{HI}$, $OPT(I) \leq OPT(I_{HI})$ and $SIZE(I) \leq SIZE(I_{HI})$.

Now observe that $I_{HI} = H_1 I_{LO}$. Given any packing of I_{LO}, we can obtain a packing of I_{HI} which uses at most k extra bins. (Just pack each item in H_1 in a separate bin.) This implies that $OPT(I_{HI}) \leq OPT(I_{LO}) + k$ and $SIZE(I_{HI}) = SIZE(I_{LO}) + SIZE(H_1) \leq SIZE(I_{LO}) + k$. □

It is worth noting that the result presented in this lemma is constructive. It is possible in $O(n \log n)$ time to construct the instances I_{LO} and I_{HI}, and given an optimal packing of I_{LO} it is possible to construct a packing of I that meets the guarantee of the above lemma. To construct I_{LO} and I_{HI}, it is necessary only to sort the items and perform the linear grouping. (Actually, one ingredient is still unspecified, namely the value of k; this will be given in the next section.) Given a packing of I_{LO}, we can assign all elements in $I \backslash G_1$ to bins according to the assignments of the corresponding members of I_{LO}; finally, each member of G_1 can get its own bin.

11.3.4 APTAS for Bin Packing

We now put together all these ingredients and obtain the APTAS. The algorithm A_ϵ, for any $\epsilon \in (0, 1]$, takes as input an instance I of BIN PACKING consisting of n items.

Algorithm A_ϵ:
Input: Instance I consisting of n item sizes $\{s_1, \ldots, s_n\}$.
Output: A packing into unit-sized bins.

1. $\delta \leftarrow \frac{\epsilon}{2}$
2. Set aside all items of size smaller than δ, obtaining an instance J of $RBP[\delta, n']$ with $n' = |J|$.
3. $k \leftarrow \lceil \frac{\epsilon^2}{2} n' \rceil$
4. Perform linear grouping on J with parameter k. Let J_{LO} be the resulting instance of $RBP[\delta, m]$ and $J_{HI} = H_1 \cup J_{LO}$, with $|H_1| = k$ and $m = \lfloor \frac{n'}{k} \rfloor$.
5. Pack J_{LO} optimally using Lenstra's algorithm on $ILP(J_{LO})$.
6. Pack the k items in H_1 into at most k bins.
7. Obtain a packing of J using the same number of bins as in steps 5 and 6, by replacing each item in J_{HI} by the corresponding (smaller) item in J.
8. Using FF, pack all the small items set aside in step 2, using new bins only if necessary.

How many bins does A_ϵ use in the worst case? Observe that we have packed the items in J_{HI}, hence the items in J, into at most $OPT(J_{LO}) + k$ bins. Consider now the value of k in terms of the optimal solution. Since all items have size at least $\epsilon/2$ in J, it must be the case that $SIZE(J) \geq \epsilon n'/2$. This implies that

$$k \leq \frac{\epsilon^2 n'}{2} + 1 \leq \epsilon \cdot SIZE(J) + 1 \leq \epsilon \cdot OPT(J) + 1$$

Using Lemma 11.7, we obtain that J is packed into a number of bins not exceeding

$$OPT(J_{LO}) + k \leq OPT(J) + \epsilon \cdot OPT(J) + 1 \leq (1 + \epsilon) \cdot OPT(J) + 1$$

Finally, Lemma 11.6 implies that, while packing the small items at the last step, we use a number of bins not exceeding

$$\max\{(1 + \epsilon) \cdot OPT(J) + 1,\ (1 + \epsilon) \cdot OPT(I) + 1\} \leq (1 + \epsilon) \cdot OPT(I) + 1$$

since $OPT(J) \leq OPT(I)$. We have obtained the following theorem.

Theorem 11.2

The algorithm A_ϵ finds a packing of I into at most $(1 + \epsilon) \cdot OPT(I) + 1$ bins in time $c(\epsilon) n \log n$, where $c(\epsilon)$ is a constant depending only on ϵ.

For the running time, note that the only really expensive step in the algorithm is the one where we solve ILP using Lenstra's algorithm. As we observed earlier, this requires time linear in n, although it may be severely exponential in δ and m, which are functions of ϵ.

11.4 Asymptotic Fully Polynomial-Time Approximation Scheme

Our next goal is to convert the preceding APTAS into an AFPTAS. The reason that the above scheme is not fully polynomial is the use of the algorithm for integer linear programming, which requires time exponential in $1/\epsilon$. We now describe a technique for getting rid of this step via the construction of a "fractional" solution to the restricted bin packing problem, and a "rounding" of this to a feasible solution which is not very far from optimal. This is based on the ideas due to Karmakar and Karp.

11.4.1 Fractional Bin Packing and Rounding

Consider again the problem *RBP*$[\delta, m]$. By the preceding discussion, any instance I of this problem can be formulated as the integer linear program *ILP*(I).

$$\begin{array}{ll} \text{minimize} & \vec{x} \cdot \vec{1} \\ \text{subject to} & \\ & \vec{x} \geq 0 \\ & \vec{x} \cdot A = \vec{n} \end{array}$$

Note that we are stating the last constraint as we originally did: as an equality. Recall that A is a $q \times m$ matrix, $\vec{x}$ a q-vector, and $\vec{n}$ an m-vector. The bin types matrix A as well as $\vec{n}$ are determined by the instance I.

Consider now the linear programming relaxation of *ILP*(I). This system *LP*(I) is exactly the same as *ILP*(I), except that we now relax the requirement that $\vec{x}$ be an integer vector. Recall that *SIZE*(I) is the total size of the items in I, and that *OPT*(I) is the value of the optimal solution to *ILP*(I) as well as the smallest number of bins into which the items of I can be packed.

Definition 11.13

LIN(I) *is the value of the optimal solution to LP*(I)*, the linear programming relaxation of ILP*(I).

What does a noninteger solution to *LP*(I) mean? The value of x_i is a real number that denotes the number of bins of type T^i which are used in the optimal packing. One may interpret this as saying that items can be "broken up" into fractional parts, and these fractional parts can then be packed into fractional bins. This in general would give us a solution of value *SIZE*(I), but keep in mind that the constraints in *LP*(I) do not allow any arbitrary "fractionalization." The constraints require that in any fractional bin, the items packed therein must be the same fraction of the original items. Thus, this solution does capture some of the features of the original problem. We will refer to the solution of *LP*(I) as a *fractional bin packing.*

To analyze the relationship between the fractional and integral solutions to any instance we will have to use some basic facts from the theory of linear programming. The uninitiated reader is referred to any standard textbook for a more complete treatment; e.g., see the book by Papadimitriou and Steiglitz [8].

Consider the system of linear equations implicit in the constraint[2] $\vec{x}.A = \vec{n}$. Here we have m linear equations in q variables, where q is much larger than m. This is an underconstrained system of equations. Let us assume that $rank(A) = m$; it is easy to modify the following analysis when $rank(A) < m$. Assume, without loss of generality, that the first m rows of A form a basis, i.e., they are linearly independent. The following are standard observations from linear programming theory.

Definition 11.14

A basic feasible solution *to LP is a solution* $\vec{x}^*$ *such that only the entries corresponding to the basis of A are nonzero. In other words,* $x_i^* = 0$ *for all* $i > m$.

Fact 11.3

Every LP has an optimal *solution which is a basic feasible solution.*

We can now derive the following lemma which relates *LIN*(I) to both *SIZE*(I) and *OPT*(I).

Lemma 11.8

For all instances I of RBP$[\delta, m]$,

$$SIZE(I) \leq LIN(I) \leq OPT(I) \leq LIN(I) + \frac{m+1}{2}$$

[2] We will ignore the nonnegativity constraints for now as they do not bear upon the following discussion.

Proof

To prove the first inequality, we note that $SIZE(I) = \sum_{j=1}^{m} n_j v_j = \sum_{j=1}^{m} (\vec{x} \cdot A_j) v_j$, where we use A_j to mean the jth column of A. This sum is equal to $\sum_{i=1}^{q} x_i (\sum_{j=1}^{m} a_{ij} v_j)$. Note that for all $1 \leq i \leq q$, $\sum_{j=1}^{m} a_{ij} v_j \leq 1$ is the total size accounted for by the ith bin type and is therefore at most 1. It follows that $SIZE(I) \leq \sum_{i=1}^{q} x_i = LIN(I)$. The second inequality follows from the observation that an optimal solution to $ILP(I)$ is also a feasible solution to $LP(I)$.

To see the last inequality, fix I and let $\vec{y}$ be some basic optimal solution to $LP(I)$. Since $\vec{y}$ has at most m nonzero entries, it uses only m different types of bins. Rounding up the value of each component of $\vec{y}$ will increase the number of bins by at most m, and will yield a solution to ILP. The bound promised in the lemma is slightly stronger and may be observed as follows. Define the vectors $\vec{w}$ and $\vec{z}$ in the following way:

$$\forall i, \quad w_i = \lfloor y_i \rfloor$$
$$\forall i, \quad z_i = y_i - w_i$$

The vector $\vec{w}$ is the integer part of the solution and $\vec{z}$ the fractional part. Let J denote the instance of $RBP[\delta, m]$ that consists of the items not packed in the (integral) solution specified by $\vec{w}$. (Note that J is, indeed, a legal instance of $RBP[\delta, m]$, i.e., all items occur in integral quantities, because in $\vec{w}$, all bin types, and therefore all items, occur in integral quantities.) The vector $\vec{z}$ gives a fractional packing of the items in J, such that each of the m bin types is used a number of times which is a fraction less than 1.

Just as $SIZE(I) \leq LIN(I)$, a similar argument implies that

$$SIZE(J) \leq LIN(J)$$

By Lemma 11.2 we know that

$$OPT(J) \leq 2 \cdot SIZE(J) + 1$$

It is also obvious that $OPT(J) \leq \sum_{i=1}^{m} z_i \leq m$, since rounding each nonzero z_i up to 1 gives a feasible packing of J. Thus,

$$\begin{aligned} OPT(J) &\leq \min\{m,\ 2 \cdot SIZE(J) + 1\} \\ &\leq (m + 2 \cdot SIZE(J) + 1)/2 \\ &= SIZE(J) + \frac{m+1}{2} \end{aligned}$$

We will now bound $OPT(I)$ in terms of $LIN(I)$ and m.

$$\begin{aligned} OPT(I) &\leq OPT(I - J) + OPT(J) \\ &\leq \sum_{i=1}^{m} w_i + \left(SIZE(J) + \frac{m+1}{2} \right) \\ &\leq \sum_{i=1}^{m} w_i + LIN(J) + \frac{m+1}{2} \\ &\leq \sum_{i=1}^{m} w_i + \sum_{i=1}^{m} z_i + \frac{m+1}{2} \\ &= LIN(I) + \frac{m+1}{2} \end{aligned}$$

The first inequality follows from the fact that independent integer packings of $I - J$ and J can be combined to form an integer packing of I. The second and third follow from facts proved above, and the fact that $\vec{w}$ is a feasible solution to the $RBP[\delta, m]$ instance I. The fourth holds because $\vec{z}$ is a feasible fractional packing of J. Finally, the equality holds by the optimality of $\vec{y}$ as a solution to $LIN(I)$. □

It is not very hard to see that all of the above is constructive. More precisely, given the solution to $LP(I)$, we can construct in linear time a solution to I such that the bound from the above theorem is met: We take an optimal basic solution $\vec{y}$ and break it into $\vec{w}$ and $\vec{z}$ as described, and define J as above. We find an integral solution for J either by rounding up each nonzero entry to 1 or by using First Fit, whichever produces a better solution. We then put together the solution given by $\vec{w}$ and that found for J.

The only problem is that it is not obvious that we can solve the linear program in fully polynomial time, even though there exist polynomial-time algorithms for linear programming [9], unlike the general problem of integer programming. The reason is that the number of variables is still exponential in $1/\epsilon$. All we have achieved is that we no longer need to solve an integer program.

Karmakar and Karp show how to get around this problem by resorting to the ellipsoid method of Grötschel et al. [6,7,10]. In this method, it is possible to solve a linear program with an exponential number of constraints in time which is polynomial in the number of variables and the number sizes, given a *separation oracle*. A separation oracle takes any proposed solution vector $\vec{x}$ and either guarantees that it is a feasible solution, or provides any one constraint which is violated by it. Karmakar and Karp gave an efficient construction of a separation oracle for $LP(I)$. This would result in a polynomial-time algorithm for $LP(I)$ if it had a small number of variables, even if it has an exponential number of constraints. Since our situation is exactly the reverse, i.e., we have a small number of constraints and an exponential number of variables, we will consider the *dual linear program* for $LP(I)$, which has the desired features of a small number of variables. By Linear Program Duality, its optimal solution corresponds exactly to the optimal solution of $LP(I)$.

One important detail is that it is impossible to solve $LP(I)$ exactly in fully polynomial time. However, it can be solved within an additive error of 1 in fully polynomial time. Moreover, the implementation of the separation oracle is in itself an approximation algorithm. The idea behind this method is due to Gilmore and Gomory [11] who observed that, in the case of an infeasible proposed solution, a violated constraint can be computed via the solution of a knapsack problem. Since this problem is $\mathcal{NP}$-complete, one must resort to the use of an approximation scheme for KNAPSACK [3], and so the solution of the dual is not exact but a close approximation. Karmakar and Karp used this approximate solution to the dual to obtain an approximate lower bound on the optimal value of the original problem. Having devised the procedure for efficiently computing an approximate lower bound, they then construct an approximate solution.

This algorithm is rather formidable and the details are omitted as it is outside the scope of this discussion. The following theorem results.

Theorem 11.3

There is a fully polynomial-time algorithm A for solving an instance I of $RBP[\delta, m]$ such that $A(I) \leq LIN(I) + \frac{m+1}{2} + 1$.

11.4.2 AFPTAS for Bin Packing

We are now ready to present the AFPTAS for BIN PACKING. We will need the following variant of Lemma 11.7.

Lemma 11.9

Using the linear grouping scheme on an instance I of $RBP[\delta, n]$, we obtain an instance I_{LO} of $RBP[\delta, m]$ and a group H_1 such that, for $I_{HI} = H_1 I_{LO}$,

$$LIN(I_{LO}) \leq LIN(I) \leq LIN(I_{HI}) \leq LIN(I_{LO}) + k$$

Proof

The proof is almost identical to that of Lemma 11.7. Recall that $m = \lfloor n/k \rfloor$. Take the original instance I, and define $G_1, \ldots, G_{m+1}, H_1, \ldots, H_m, I_{LO}$, and I_{HI} as before. From Lemma 11.3 the first two inequalities follow. The third follows from the fact that, given a solution to I_{LO}, we can solve I_{HI} by putting all members of $I_{HI} \cup I_{LO}$ in the bins assigned by the given solution, and then putting each member of H_1 in a bin by itself. □

The basic idea behind the AFPTAS of Karmakar and Karp is very similar to that used in the APTAS. We first eliminate all the small items, and then apply linear grouping to the remaining items. The resulting instance of $RBP[\delta, m]$ is then formulated as an *ILP*, and the solution to the corresponding relaxation *LP* is computed using the ellipsoid method. The fractional solution is then rounded to an integer solution. The small items are then added into the resulting packing exactly as before.

Algorithm A_ϵ:
Input: Instance I consisting of n item sizes $\{s_1, \ldots, s_n\}$.
Output: A packing into unit-sized bins.

```
1. $\delta \leftarrow \frac{\epsilon}{2}$.
2. Set aside all items of size smaller than $\delta$, obtaining an instance $J$ of $RBP[\delta, n']$ with $n' = |J|$.
3. $k \leftarrow \lceil \frac{\epsilon^2 n'}{2} \rceil$
4. Perform linear grouping on $J$ with parameter $k$. Let $J_{LO}$ be the resulting instance of $RBP[\delta, m]$ and
   $J_{HI} = H_1 \cup J_{LO}$, with $|H_1| = k$ and $m = \lfloor \frac{n'}{k} \rfloor$.
5. Pack the $k$ items in $H_1$ into at most $k$ bins.
6. Pack $J_{LO}$ using the ellipsoid method and rounding the resulting fractional solution.
7. Obtain a packing of $J$ using the same number of bins as used for $J_{HI}$, by replacing each item in $J_{HI}$
   by the corresponding (smaller) item in $J$.
8. Using FF, pack all the small items set aside in step 2, using new bins only if necessary.
```

Theorem 11.4

The approximation scheme $\{A_\epsilon : \epsilon > 0\}$ is an AFPTAS for BIN PACKING *such that*

$$A_\epsilon(I) \leq (1+\epsilon) \cdot OPT(I) + \frac{1}{\epsilon^2} + 3$$

Proof
The running time is dominated by the time required to solve the linear program, and we are guaranteed that this is fully polynomial.

By Lemma 11.8, the number of bins used to pack the items in J_{LO} is at most

$$(LIN(J_{LO}) + 1) + \frac{m+1}{2} \leq OPT(I) + \frac{1}{\epsilon^2} + 2$$

given the preceding lemmas and the choice of m. The number of bins used to pack the items in H_1 is at most k, which in turn can be bounded as follows using the observation that $OPT(J) \geq SIZE(J) \geq \epsilon n'/2$:

$$k \leq \left\lceil \frac{n'\epsilon^2}{2} \right\rceil \leq \epsilon \cdot OPT(J) + 1 \leq \epsilon \cdot OPT(I) + 1$$

Thus, the total number of bins used to pack the items in J cannot exceed

$$(1+\epsilon) \cdot OPT(I) + \frac{1}{\epsilon^2} + 3$$

Lemma 11.6 guarantees that the small items can be added without an increase in the number of bins, and so the desired result follows. □

11.5 Related Results

We conclude the chapter by presenting a literature survey on topics related to BIN PACKING and asymptotic approximation schemes.

BIN PACKING is a classic problem in theoretical computer science; the algorithms proposed for this problem, and the analysis of these algorithms, employ a wide variety of techniques. In the foregoing discussion, we used the fact that the First Fit algorithm has an asymptotic worst-case performance ratio of 2, but this is not the best bound. Ullman [12] proved an asymptotic worst-case performance bound of 17/10 for this algorithm, and subsequent papers [13–15] reduced the additive constant term from 3 to 1 or less eventually. First Fit is not the only algorithm considered for BIN PACKING. Many other online algorithms, semionline algorithms, and offline algorithms have been proposed and their worst- and average-case behavior studied extensively. We refer the reader to survey articles by Coffman et al. [16–18], and Chapters 32–35 of this handbook for further details.

There are several commonly considered variants of the basic bin packing problem, all of which are $\mathcal{NP}$-complete. In most of these cases, it is reasonably easy to come up with bounded-ratio approximations. These variants can be classified under four main headings: packings in which the number of items per bin is bounded, packings in which certain items cannot be packed into the same bin, packings in which there are constraints (e.g., partial orders) on the way in which the items are packed, and dynamic packings in which items may be added and deleted. These variants are discussed in Chapters 33–35 of this handbook.

There are also some generalizations of the basic packing problem, many of which are covered in the three survey papers and chapters mentioned above. While some generalizations do not admit APTAS or AFPTAS, several approximation schemes have been found successful, generally based on the ideas described above. Here we focus on three generalizations that admit APTAS and AFPTAS: packings into variable-sized bins, multidimensional bin packing, and BIN COVERING, the dual of BIN PACKING.

Murgolo shows an approximation scheme for the case of variable-sized bins [19]. For multidimensional bin packing, APTAS have recently been found for packing d-dimensional[3] cubes into the minimum number of unit cubes by Bansal and Sviridenko [20], and Correa and Kenyon [21], independently. Interestingly, the problem of packing (two-dimensional) rectangles into squares does not admit APTAS or AFPTAS [20]. However, for a more restricted version, namely, a two-stage packing of the rectangles, Caprara et al. show an AFPTAS [22]. The dual problem of BIN PACKING is BIN COVERING, in which we want to maximize the total number of bins used, but must fill each bin to at least a certain capacity. Jansen and Solis-Oba show an AFPTAS for BIN COVERING [23].

BIN PACKING is not the only problem that admits APTAS and AFPTAS. Raghavan and Thompson give an APTAS for the 0–1 multicommodity flow problem [24]. Their approaches include probabilistic rounding of fractional linear-programming solutions. Cohen and Nakibli [25] show an APTAS for a somewhat related problem, the n-hub shortest path routing problem. The goal is to minimize the overloading of links in a directed network with pairwise source–sink flows, by setting an n-hub route for each source–sink pair. This APTAS also uses probabilistic rounding. Aingworth et al. [26] show an AFPTAS for pricing Asian options on the lattice, using discretization to reduce the number of possible option values.

There are other problems that admit *absolute* approximation algorithms, i.e., algorithms guaranteed to produce solutions whose costs are at most an additive constant away from the optimal. In contrast to APTAS and AFPTAS, whose approximation ratios approach a value arbitrarily *close* to 1 as the optimal cost grows, these algorithms have an asymptotic performance ratio *equal* to 1; that is, as the optimal cost grows, the approximation ratio of an absolute approximation algorithm approaches 1 itself. Examples of problems admitting absolute approximations include minimum-edge coloring [27] and minimum-degree spanning tree [28], where the approximate solution is guaranteed to exceed the optimal solution by at most 1. The techniques used in these algorithms, however, differ from the ones discussed in this chapter. A variation of Karmakar and Karp's ideas leads to a stronger result for BIN PACKING, which is the construction of an approximation algorithm A that is fully polynomial and has the performance guarantee $A(I) \leq OPT(I) + O(\log^2 OPT(I))$. One is tempted to believe that there also exists an absolute approximation algorithm for BIN PACKING, i.e., an algorithm that runs in polynomial time and guarantees that $A(I) \leq OPT(I) + O(1)$. The existence of such an algorithm is still an open question.

[3] Here d is assumed to be a fixed constant.

References

[1] Fernandez de la Vega, W. and Lueker, G. S., Bin packing can be solved within $1 + \epsilon$ in linear time, *Combinatorica,* 1, 349, 1981.

[2] Karmakar, N. and Karp, R. M., An efficient approximation scheme for the one-dimensional bin packing problem, *Proc. FOCS,* 1982, p. 312.

[3] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., New York, 1979.

[4] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M., *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, Springer, Berlin, 1999.

[5] Lenstra, H. W., Integer programming with a fixed number of variables, *Math. Oper. Res.,* 8, 538, 1983.

[6] Grötschel, M., Lovász, L., and Schrijver, A., *Geometric Algorithms and Combinatorial Optimization*, Springer, Berlin, 1987.

[7] Schrijver, A., *Theory of Linear and Integer Programming*, Wiley, New York, 1986.

[8] Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity,* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[9] Karmakar, N., A new polynomial-time algorithm for linear programming, *Combinatorica,* 4, 373, 1984.

[10] Grötschel, M., Lovász, L., and Schrijver, A., The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica,* 1, 169, 1981.

[11] Gilmore, P. C. and Gomory, R. E., A linear programming approach to the cutting-stock problem, *Oper. Res.,* 9, 849, 1961.

[12] Ullman, J. D., The Performance of a Memory Allocation Algorithm. Technical report 100, Princeton University, Princeton, NJ, 1971.

[13] Garey, M. R., Graham, R. L., and Ullman, J. D., Worst-case analysis of memory allocation algorithms, *Proc. of STOC*, 1972, p. 143.

[14] Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., and Graham, R. L., Worst-case performance bounds for simple one-dimensional packing algorithms, *SIAM J. Comput.,* 3, 299, 1974.

[15] Garey, M. R., Graham, R. L., Johnson, D. S., and Yao, A. C., Resource constrained scheduling as generalized bin packing, *J. Comb. Theory, Ser. A,* 21, 257, 1976.

[16] Coffman, E. G., Garey, M. R., and Johnson, D. S., Approximation algorithms for bin packing—an updated survey, in *Algorithm Design for Computer System Design*, Ausiello, G., Lucertini, M., and Serafini, P., Eds., Springer, Berlin, 1984.

[17] Coffman, E. G., Garey, M. R., and Johnson, D. S., Approximation algorithms for bin packing—a survey, in *Approximation Algorithms for NP-Hard Problems*, Hochbaum, D. S., Ed., PWS Publishing Co., Boston, MA, 1996.

[18] Coffman, E. G., Csirik, J., and Woeginger, G. J., Approximate Solutions to Bin Packing Problems, Technical report Woe-29, Institut für Mathematik B, TU Graz, Steyrergasse 30, A-8010 Graz, Austria, 1999.

[19] Murgolo, F. D., An efficient approximation scheme for variable-sized bin packing, *SIAM J. Comput.,* 16, 149, 1987.

[20] Bansal, N. and Sviridenko, M., New approximability and inapproximability results for 2-dimensional bin packing, *Proc. of SODA,* 2004, p. 196.

[21] Correa, J. R. and Kenyon, C., Approximation schemes for multidimensional packing, *Proc. of SODA,* 2004, p. 186.

[22] Caprara, A., Lodi, A., and Monaci, M., Fast approximation schemes for two-stage, two-dimensional bin packing, *Math. Oper. Res.,* 30, 150, 2005.

[23] Jansen, K. and Solis-Oba, R., An asymptotic fully polynomial time approximation scheme for bin covering, *Theor. Comput. Sci.,* 306, 543, 2003.

[24] Raghavan, P. and Thompson, C. D., Randomized rounding: a technique for provably good algorithms and algorithmic proofs, *Combinatorica,* 7, 365, 1987.

[25] Cohen, R. and Nakibli, G., On the computational complexity and effectiveness of N-hub shortest-path routing, *Proc. of the 23rd Conf. of the IEEE Communications Society*, 2004, p. 694.

[26] Aingworth, D., Motwani, R., and Oldham, J., Accurate approximations for Asian options, *Proc. of SODA,* 2000, p. 891.

[27] Vizing, V. G., On an estimate of the chromatic class of a p-graph (in Russian), *Diskret. Analiz.,* 3, 25, 1964.

[28] Fürer, M. and Raghavachari, B., Approximating the minimum degree spanning tree to within one from the optimal degree, *J. Algorithms,* 17, 409, 1994.

12

Randomized Approximation Techniques

Sotiris Nikoletseas
University of Patras and CTI

Paul Spirakis
University of Patras and CTI

12.1 Introduction

Randomization (i.e., the use of random choice as an algorithmic step) is one of the most interesting tools in designing efficient algorithms. A remarkable property of randomized algorithms is their structural simplicity. In fact, in several cases, while the known deterministic algorithms are quite involved, the randomized ones are simpler and much easier to code.

This happens also in the case of approximation algorithms to NP-hard problems. In fact, it is exactly the area of efficient approximations where the value of randomization has been demonstrated via at least two very general techniques. The first of them is the notorious randomized rounding method, which provides an unexpected association between the optimal solutions of 0/1 integer linear programs (ILPs) and their linear programming (LP) relaxations. Randomized rounding is a way to return from the fractional optimal values of the LP relaxation (which can be efficiently computed) to a good integral solution, whose expected cost is the cost of the fractional solution! We demonstrate this method here via an example application to the optimization version of the NP-complete set cover problem, and we comment also on its use (as a random projection technique) in approximations via semidefinite programs.

The second technique is used in approximately counting the number of solutions to #P-complete problems. Most of this technique is built around the Markov chain Monte Carlo (MCMC) method. It essentially states that the time required for a Markov chain to mix (to approach its steady state) is an approximate estimator of the size of the state space of the chain (i.e., the number of the combinatorial objects that we wish to count). If the Markov chain is *rapidly mixing* (i.e., it converges in polynomial time), then we can also count the size of the state space approximately in polynomial time. We demonstrate this second approach here via an application to approximate counting of a special kind of colorings of the vertices of a graph.

The main drawback of the use of randomization in approximations is that it may only derive good results in expectation (or sometimes with high probability). This means that in certain inputs a randomized approximation technique may take a lot of time (if we want it not to fail) or may even fail. In certain cases, it is possible to convert a randomized approach to a deterministic one via a *derandomization* technique (for example, either by making the random choices dependent on each other and thus reduce the amount of randomness to the limit of allowing a deterministic brute-force search of the probability space, or by the use of *conditional probabilities*). We do not discuss derandomization here, since its application has been quite limited and since our purpose is to let the reader appreciate the simplicity and generality of the randomized methods.

12.2 Optimization and Randomized Rounding

12.2.1 Introduction

NP-hard optimization problems are not known to allow finding optimal solutions efficiently. Their combinatorial structure is elaborate and sometimes quite cryptic in the general sense.

Many NP-hard optimization problems can be coded as ILPs. In fact, in quite a lot of them, the values of the integer variables involved are only 0 and 1. We are then speaking of 0/1 Integer Linear Programming Problems (or 0/1 ILP). An example here is hard problems involving Boolean solutions.

Relaxing the integer constraints of the form "$x_i \in \{0, 1\}$" to the linear inequalities "$0 \leq x_i \leq 1$," converts a 0/1 ILP into an LP problem. Nowadays it is known that LP optimization problems can be solved in polynomial time (via the ellipsoid or interior point methods). This strong similarity between 0/1 ILP and LP allows us to design efficient approximation algorithms for the hard problem at hand.

A feasible solution to the LP-relaxation can be thought of as a *fractional solution* to the original problem. The set of feasible solutions of a system of linear inequalities is known to build a *polytope* (a convex, multidimensional object, a *polyhedron*, like a diamond). To search for an optimum with respect to a linear function in a polytope is not so hard, since it has been proved that the optimum is located in some vertex of the polytope. However, in the case of an NP-hard problem, we cannot expect the polyhedron defining the set of feasible solutions to have integer vertices. Thus, our task is to somehow transform the optimal solution of the LP relaxation into a near-optimal *integer* solution.

A basic technique for obtaining approximation algorithms using LP is what we call *LP rounding*: i.e., solve the (relaxed) linear program and then convert the fractional solutions obtained (e.g., $x_i = 2/3$) to an integral solution (e.g., here $x_i = 1$ seems more reasonable than $x_i = 0$) trying of course to make sure that the cost of the solution does not increase much in the process.

A natural idea for rounding an optimal fractional solution is to view the fractions as *probabilities*. Then we can "flip coins" with these probabilities as biases, and round accordingly. So, the case "$x_i = 2/3$," obtained via LP, now leads to an experiment where "$x_i = 1$ with probability 2/3 and 0 else." This idea is called *randomized rounding*. In the sequel, we present the method via an application to the set cover problem. This application is also demonstrated in the book of Vazirani [1]. We try to be more thorough here and provide details.

12.2.2 The Set Cover Problem

The set cover problem is one of the oldest known NP-complete problems (it generalizes the vertex cover problem).

Problem SET COVER

Given is a universal set U of n elements and also a collection of subsets of U, $S = \{S_1, S_2, \ldots, S_k\}$. Given is also a cost function $c : S \to Q^+$.

We seek to find a minimum cost subcollection of S that covers all the elements of U.

Note here that the cost of a subcollection of S, e.g., $F = \{S_{i_1}, \ldots, S_{i_\lambda}\}$ is $\sum_{j=1}^{\lambda} c(S_{i_j})$. Note also that any feasible answer to the problem requires covering all of U, i.e., if F is a feasible answer, then we demand that $\bigcup_{j=1}^{\lambda} S_{i_j} = U$. Define the *frequency* of an element of U to be the number of sets it is in.

Let us denote the frequency of the most frequent element by f. The various known approximation algorithms for set cover achieve one of the two approximation factors $O(\log n)$ or f. The special case with $f = 2$ is, basically, the vertex cover problem in graphs (see Ref. [2]).

12.2.3 The Set Cover as an Integer Program

To formulate the set cover problem as a problem in 0/1 ILP, let us assign a variable $x(S_i)$ for each set $S_i \in S$. This variable will have the value 1 iff the set S_i is selected to be in the set cover (and will have the value 0 else).

Clearly, for each element $\alpha \in U$ we want it to be covered, i.e., we want it to be in at least one of the picked sets. In other words, we want, for each $\alpha \in U$ that, at least one of the sets containing it is picked by our candidate algorithm. These considerations give the following ILP program.

Set Cover ILP

Minimize $\sum_{S_i \in S} c(S_i)x(S_i)$

subject to:

$\forall \alpha \in U \;\; \sum_{S_i : \alpha \in S_i} x(S_i) \geq 1$

and $x(S_i) \in \{0, 1\}$ for all $S_i \in S$.

The LP relaxation of this integer program can be obtained by replacing each "$x(S_i) \in \{0, 1\}$" with "$0 \leq x(S_i) \leq 1$." The reader can easily see that the upper bound on $x(S_i)$ is redundant here. So we get the following linear program:

Set Cover Relaxation

Minimize $\sum_{S_i \in S} c(S_i)x(S_i)$

subject to:

(1) $\forall \alpha \in U \;\; \sum_{S_i : \alpha \in S_i} x(S_i) \geq 1$

(2) $\forall S_i \in S \;\; x(S_i) \geq 0$

Note 1: A solution to the above LP is a "fractional" set cover.

Note 2: A fractional set cover may be *cheaper* than the optimal (integral) set cover! To see this, let $U = \{\alpha_1, \alpha_2, \alpha_3\}$ and $S = \{S_1, S_2, S_3\}$ with $S_1 = \{\alpha_1, \alpha_3\}$, $S_2 = \{\alpha_2, \alpha_3\}$ and $S_3 = \{\alpha_3, \alpha_1\}$, and let $c(S_i) = 1$, $i = 1, 2, 3$. Any integral cover must pick two sets, for a cost of 2. However, if we pick each set with $x(S_i) = 1/2$, we satisfy all the constraints and the cost of the fractional cover obtained is $3/2$.

Note 3: The LP dual (see, e.g., Ref. [3]) of the set cover relaxation is a "packing" LP: The dual tries to pack the "material" into elements, trying to maximize the total amount packed, but no set must be "overpacked" (i.e., the total amount of the material packed into the elements of the set should not exceed its cost). The duality of covering–packing is a basic remark and has given lots of approximation results.

12.2.4 A Randomized Rounding Approximation to Set Cover

Let $x(S_i) = p_i$, $i = 1, \ldots, k$ be an *optimal* solution to the set cover relaxation program. Such a solution can be found in polynomial time.

Now, for each $S_i \in S$, select S_i with probability p_i, *independently* of the other selections.

Note: We can do it via choosing (independently for each i) k values $\gamma_1, \ldots, \gamma_k$ randomly and uniformly from the interval $[0, 1]$. Then, for $i = 1$ to k, if $\gamma_i \in [0, p_i]$, we select S_i, otherwise we do not.

Let F be the collection of the selected sets via the experiment.

The expected cost of F is

$$E(cost(F)) = \sum_{S_i \in S} c(S_i) \cdot Prob\{S_i \text{ is selected}\}$$

that is,

$$E(cost(F)) = \sum_{S_i \in S} c(S_i)\, p_i$$

But $\{p_i,\ i = 1, \dots, k\}$ is the optimal solution to the set cover relaxation program, hence $\sum_{S_i \in S} c(S_i)\,p_i$ is the optimal (minimum) value. Let us denote it by OPT_R (optimal for the relaxation).

Now let us examine whether F is a cover. For an $\alpha \in U$, suppose that α occurs in λ sets of S. W.l.o.g., let the probabilities of these sets in the optimal solution of the relaxation be $p_1, \dots, p_\lambda$. Since all the constraints are satisfied by the optimal solution, we get

$$p_1 + \cdots + p_\lambda \geq 1 \tag{12.1}$$

But

$$Prob\{\alpha \text{ is covered by } F\} = 1 - \prod_{i=1}^{\lambda}(1 - p_i)$$

Because of Eq. (12.1), the above expression becomes minimum when $p_i = \cdots = p_\lambda = \frac{1}{\lambda}$, so

$$Prob\{\alpha \text{ is covered by } F\} \geq 1 - \left(1 - \frac{1}{\lambda}\right)^{\lambda} \geq 1 - \frac{1}{e}$$

where $e \simeq 2.73$ is the basis of the natural logarithms. The above analysis holds for any $\alpha \in U$.

We now repeat the part of the experiment where we pick the collection F, independently each time.

Let us pick the collections $F_1, F_2, \dots, F_t$. Let $\tilde{F} = \cup_{i=1}^{t} F_i$. So, for all $\alpha \in U$

$$Prob\{\alpha \text{ is not covered by } \tilde{F}\} \leq \left(\frac{1}{e}\right)^{t}$$

By summing over all $\alpha \in U$ we have

$$Prob\{\tilde{F} \text{ is not a cover}\} \leq n\left(\frac{1}{e}\right)^{t} \tag{12.2}$$

By selecting now $t = \log \xi n$ (with $\xi \geq 4$, a constant) we eventually get

$$Prob\{\tilde{F} \text{ is not a cover}\} \leq n\frac{1}{4n} = \frac{1}{4} \tag{12.3}$$

Having established that $\tilde{F}$ is a cover with constant probability let us see its cost. Clearly,

$$E(c(\tilde{F})) \leq OPT_R \cdot \log \xi n$$

Thus, by the Markov inequality ($Prob\{X \geq mE[X]\} \leq 1/m$, for $x \geq 0$) we get

$$Prob\{c(\tilde{F}) \geq 4OPT_R \cdot \log \xi n\} \leq \frac{1}{4} \tag{12.4}$$

Let A be the (undesirable) event: "$\tilde{F}$ is not a valid cover or $c(\tilde{F})$ is at least $4OPT_R \log \xi n$."

$$Prob(A) \leq \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \tag{12.5}$$

Note that, given $\tilde{F}$, we can verify in polynomial time whether the negation of A holds. If it holds (this happens with probability $\geq 1/2$) then we have an $\tilde{F}$ which is (a) a valid set cover, (b) with cost at most $4 \log \xi n$ times above the OPT_R.

Let OPT be the optimal cost of the integer program. Clearly $OPT_R \leq OPT$ hence, when $\overline{A}$ holds, we have found a valid cover with an approximation ratio (w.r.t. the cost)

$$R = \frac{c(\tilde{F})}{OPT} \leq \frac{c(\tilde{F})}{OPT_R} \leq 4 \log \xi n$$

Now, if A happens to hold, then we repeat the entire algorithm. The expected number of repetitions needed to get a valid cover with $R = \Theta(\log n)$ is then at most 2.

We summarize all this in the following.

Theorem 12.1

The method of randomized rounding gives us in expected polynomial time a valid set cover with a cost approximation ratio $R = \Theta(\log n)$.

Note: The algorithm presented here never errs (since we repeat it if $\tilde{F}$ is not a valid cover of small cost). The penalty is time, but it is small since the number of repetitions follows a geometric distribution.

12.2.5 A Remark in the Analysis

In the analysis of the last section we established Eq. (12.2) namely

$$Prob\{\tilde{F} \text{ is not a cover}\} \leq n\left(\frac{1}{e}\right)^t$$

where t is the number of collections selected independently. By using $t = \xi \log n$ with $\xi \geq 2$ we get

$$Prob\{\tilde{F} \text{ is not a cover}\} \leq ne^{-2\log n} \leq \frac{1}{n}$$

with an expected cover cost of $E(c(\tilde{F})) \leq OPT_R \cdot \xi \log n$, i.e., $E(c(\tilde{F})) \leq OPT \cdot 2\log n$.

If we are satisfied with a good *expected* cost, we can stop here. We get a valid cover with probability at least $1 - \frac{1}{n}$ in one repetition and the expected cost is $\Theta(OPT \cdot \log n)$.

12.2.6 Literature Notes

For more information about set cover approximation via randomized rounding, see the excellent book by Vazirani, Chapter 14. For a more advanced randomized rounding method for set cover see Ref. [4]. A quite similar method can be applied to the MAX-SAT problem (see Ref. [1], Chapter 16 or Ref. [5], Chapter 7). Randomized rounding (actually the random projection method) has been also used together with semidefinite programming to give an efficient approximation to the MAX-CUT problem and its variations (see Ref. [1], Chapter 26) or the seminal work of Goemans and Williamson [6] who introduced the use of semidefinite programs in approximation algorithms.

12.3 Approximate Counting Using the Markov Chain Monte Carlo Method

The MCMC method is a development of the classic, well-known Monte Carlo method for approximately estimating measures and quantities whose exact computation is a difficult task. In fact, the Monte Carlo method expresses the quantity under evaluation (say x) as the expected value $x = E(X)$ of a random variable X, whose samples can be estimated efficiently. By taking the mean of a sufficiently large set of samples, an approximate estimation of the quantity of interest can be obtained.

Jerrum [7] illustrates the use of the Monte Carlo method by a simple example: the estimation of the area of the region of the unit square defined by a system of polynomial inequalities. To do this, points of the unit square are randomly uniformly sampled, i.e., a point is chosen uniformly at random (u.a.r.) and then it is tested whether it belongs to the region of interest (i.e., whether it satisfies or not all inequalities in the system). The probability that a randomly chosen point belongs to the area under investigation (i.e., the expectation of a random variable indicating whether the chosen point satisfies all inequalities in the system or not) is then an estimate of the area of the region of interest. By performing a sufficiently long sequence of such trials and taking their sample mean, an approximate estimation is obtained. More complex examples are the estimation of a size of a tree by sampling paths from its root to a leaf [8] and the estimation of the permanent of a 0,1 matrix [9].

It is, however, not always possible to get such samples of the random variable used. The Markov chain simulation can then be employed. The main idea of the MCMC method is to construct, for a random variable X, a Markov chain whose state space is (or includes) the range of X. The Markov chain constructed should be ergodic, i.e., it converges to a stationary distribution π, and this stationary distribution matches the probability distribution of the random variable X. The desired samples can then be (indirectly) obtained by simulating the Markov chain for sufficiently many steps T, from any fixed initial state, and by taking the final state reached. If T is large enough, the Markov chain gets very close to stationarity and, thus, the distribution of the samples obtained in this way is very close to the probability distribution of the random variable X; the obtained samples are thus close to perfect and the approximation error will be negligible.

The estimation of a sufficiently large time T is important for the efficiency of the simulation. In contrast to the classical theory of stochastic process that only studies the asymptotic convergence to the stationarity, the MCMC method investigates the nonasymptotic speed of convergence and thus the computational efficiency in practical applications of the simulation. The efficiency of an algorithm using the method depends on how small the number of simulation steps T is. In efficient algorithmic uses of the MCMC method with provable performance guarantees (not just heuristic applications), we require T to be small, i.e., very much smaller than the size of the state space of the simulated space. In other words, we want the Markov chain to get close to stationarity after a very short random walk on its state space. We call this time the "mixing time" of the chain and we say that an efficiently converging chain is "rapidly mixing."

Proving satisfactory upper bounds for the mixing time of the simulated Markov chain is in fact the most interesting (nontrivial) point in the application of the MCMC method. Several analytical tools have recently been devised, including the "canonical path" argument, the "conductance" argument, and the "coupling" method. We here choose to illustrate the application of the "coupling" method in a particular approximate counting problem, the problem of counting radiocolorings of a graph. For the other two methods, the reader can consult Refs. [7 and 10].

The approximate counting problem is a general computing task of estimating the number of elements in a combinatorial space. Several interesting counting problems turn out to be complete for the complexity class #P of counting problems, and thus efficient approximation techniques become essential. Furthermore, the problem of approximate counting is closely related to the problem of random sampling of combinatorial structures, i.e., generating the elements of a very large combinatorial space randomly according to some probability distribution. Combinatorial sampling problems have major computational applications, including (besides approximate counting) applications in statistical physics and in combinatorial optimization.

12.3.1 Radiocolorings of Graphs

An interesting variation of graph coloring is the k-coloring problem of graphs, defined as follows ($D(u, v)$ below denotes the distance of vertices u and v in a graph G).

Definition 12.1 k-Coloring Problem (Hale [11])

Given a graph $G(V, E)$ find a function $\phi : V \rightarrow \{1, \ldots, \infty\}$ such that $\forall\ u, v \in V, x \in \{0, 1, \ldots, k\}$: if $D(u, v) = k - x + 1$ then $|\phi_u - \phi_v| \geq x$. This function is called a k-coloring of G. Let $|\phi(V)| = \lambda$. Then λ is the number of colors *that ϕ actually uses (it is usually called* order *of G under ϕ). The number $\nu = max_{v\in V}\phi(v) - min_{u\in V}\phi(u) + 1$ is usually called the* span *of G under ϕ.*

The problem of k-coloring graphs is well motivated by practical considerations and algorithmic applications in modern networks. In fact, k-coloring is a discrete version of the frequency assignment problem (FAP) in wireless networks. Frequency assignment problem aims at assigning frequencies to transmitters exploiting frequency reuse while keeping signal interference to acceptable levels. The interference between transmitters are modeled by an interference graph $G(V, E)$, where $V(|V| = n)$ corresponds to the set of transmitters and E represents distance constraints (e.g., if two neighbor nodes in G get the same or close frequencies then this causes unacceptable levels of interference). In most real-life cases, the network topology formed has some special properties, e.g., G is a lattice network or a planar graph. The FAP is usually modeled by variations of the graph coloring problem. The set of colors represents the available frequencies. In addition, each color in a particular assignment gets an integer value which has to satisfy certain inequalities compared to the values of colors of nearby nodes in G (frequency-distance constraints). The FAP has been considered in Refs. [12–14]. *Planar* interference graphs have been studied in Refs. [15,16].

We have studied the case of k-coloring problem, where k = 2 called the radiocoloring problem (RCP).

Definition 12.2 RCP

Given a graph $G(V, E)$ find a function $\Phi : V \rightarrow N^*$ such that $|\Phi(u) - \Phi(v)| \geq 2$ if $D(u, v) = 1$ and $|\Phi(u) - \Phi(v)| \geq 1$ if $D(u, v) = 2$. The least possible number λ (order) needed to radiocolor G is denoted by $X_{order}(G)$. The least possible number $\nu = max_{v \in V}\Phi(v) - min_{u \in V}\Phi(u) + 1$ (span) needed for the radiocoloring of G is denoted as $X_{span}(G)$.

Real networks reserve bandwidth (range of frequencies) rather than distinct frequencies. In this case, an assignment seeks to use as small range of frequencies as possible. It is sometimes desirable to use as few distinct frequencies of a given bandwidth (span) as possible, since the unused frequencies are available for other use. Such optimization versions of the RCP are defined as follows.

Definition 12.3 Min Span RCP

The optimization version of the RCP that tries to minimize the span. The optimal span is called X_{span}.

Definition 12.4 Min Order RCP

The optimization version of the RCP that tries to minimize the order. The optimal order is called X_{order}.

Fotakis et al. [17] provide an $O(n\Delta)$ algorithm that *approximates* the minimum order of RCP, X_{order}, of a planar graph G *by a constant ratio which tends to 2* as the maximum degree Δ of G increases.

We study here the problem of *estimating the number of different radiocolorings* of a planar graph G. This is a #P-complete problem. We employ here standard techniques of rapidly mixing Markov chains and the *new method of coupling* for proving *rapid convergence* (see, e.g., Ref. [18]) and we present *a fully polynomial randomized approximation scheme (FPRAS)* for estimating the number of radiocolorings with λ colors for a planar graph G, when $\lambda \geq 4\Delta + 50$.

Results on radiocoloring other types (periodic, hierarchical) of graphs can be found in [19–21].

12.3.2 Outline of Our Approach

Let G be a *planar* graph of maximum degree $\Delta = \Delta(G)$ on vertex set $V = \{0, 1, \ldots, n-1\}$ and C be a set of λ colors. Let $\Phi : V \rightarrow C$ be a (proper) radiocoloring assignment of the vertices of G. Such a radiocoloring always exists if $\lambda \geq 2\Delta + 25$ and can be found by the $O(n\Delta)$ time algorithm provided in Ref. [17].

Consider the Markov chain (X_t) whose state space $R = R_\lambda(G)$ is the set of all radiocolorings of G with λ colors and whose transition probabilities from state (radiocoloring) X_t are modeled by

1. choosing a vertex $v \in V$ and a color $c \in C$ uniformly at random (u.a.r.),
2. recoloring vertex v with color c. If the resulting coloring X' is a *valid* radiocoloring assignment then let $X_{t+1} = X'$, else $X_{t+1} = X_t$.

The procedure above is similar to the "Glauber dynamics" of an antiferromagnetic Potts model at zero temperature, and was used in Ref. [18] to estimate the number of proper colorings of any low-degree graph with k colors.

The Markov chain (X_t), which we refer to in the sequel as $M(G, \lambda)$, is *ergodic* (as we show below), provided $\lambda \geq 2\Delta + 26$, in which case its stationary distribution is *uniform* over R. We show here that $M(G, \lambda)$ is *rapidly mixing*, i.e., converges, in time polynomial in n, to a close approximation of the stationary distribution, provided that $\lambda \geq 2(2\Delta + 25)$. This can be used to get an FPRAS for the number of radiocolorings of a planar graph G with λ colors, in the case where $\lambda \geq 4\Delta + 50$.

12.3.3 The Ergodicity of the Markov Chain $M(G, \lambda)$

For $t \in N$ let $P^t : R^2 \to [0, 1]$ denote the t-step transition probabilities of the Markov chain $M(G, \lambda)$ so that $P^t(x, y) = \Pr\{X_t = y | X_0 = x\}, \forall x, y \in R$. It is easy to verify that $M(G, \lambda)$ is (a) *irreducible* and (b) *aperiodic*. The irreducibility of $M(G, \lambda)$ follows from the observation that any radiocoloring x may be transformed to any other radiocoloring y by sequentially assigning new colors to the vertices V in ascending sequence; before assigning a new color c to vertex v it is necessary to recolor all vertices $u > v$ that have color c. If we assume that $\lambda \geq 2\Delta + 26$ colors are given, removing the color c from this set, we are left with $\geq 2\Delta + 25$ for the coloring of the rest of the graph. The algorithm presented in Ref. [17] shows that the remaining graph can by radiocolored with a set of colors of this size. Hence, color c can be assigned to v.

Aperiodicity follows from the fact that the loop probabilities are $P(x, x) \neq 0, \forall x \in R$.

Thus, the finite Markov chain $M(G, \lambda)$ is *ergodic*, i.e., it has a stationary distribution $\pi : R \to [0, 1]$ such that $\lim_{t \to \infty} P^t(x, y) = \pi(y), \forall x, y \in R$. Now if $\pi' : R \to [0, 1]$ is any function satisfying "local balance," i.e., $\pi'(x)P(x, y) = \pi'(y)P(y, x)$ then if $\sum_{x \in R} \pi'(x) = 1$ it follows that π' is indeed the stationary distribution. In our case $P(y, x) = P(x, y)$, thus the stationary distribution of $M(G, \lambda)$ is *uniform*.

12.3.4 Rapid Mixing

The efficiency of any approach like this to sample radiocolorings crucially depends on the rate of convergence of $M(G, \lambda)$ to stationarity. There are various ways to define closeness to stationarity but all are essentially equivalent in this case and we will use the "variation distance" at time t with respect to initial vertex x:

$$\delta_x(t) = \max_{S \subseteq R} \left| P^t(x, S) - \pi(S) \right| = \frac{1}{2} \sum_{y \in R} \left| P^t(x, y) - \pi(y) \right|$$

where $P^t(x, S) = \sum_{y \in S} P^t(x, y)$ and $\pi(S) = \sum_{x \in S} \pi(x)$.

Note that this is a *uniform bound* over all events $S \subseteq R$ of the difference of probabilities of event S under the stationary and t-step distributions.

The *rate of convergence to stationarity* from initial vertex x is

$$\tau_x(\epsilon) = \min\{t : \delta_x(t') \leq \epsilon, \forall t' \geq t\}$$

Our strategy is to use the coupling method, i.e., construct a coupling for $M = M(G, \lambda)$, i.e., a stochastic process (X_t, Y_t) on $R \times R$ such that each of the processes (X_t), (Y_t), considered in isolation, is a faithful copy of M. We will arrange a joint probability space for (X_t), (Y_t) so that, far from being independent, the two processes tend to *couple* so that $X_t = Y_t$ for t large enough. If coupling can occur rapidly (independently of the initial states X_0, Y_0), we can infer that M is rapidly mixing, because the variation distance of M from the stationary distribution is bounded above by the probability that (X_t) and (Y_t) *have not coupled* by time t.

The key result we use here is the *Coupling Lemma* (see Ref. [22] and Chapter 4 by Jerrum [7]), which apparently makes its first explicit appearance in the work of Aldous [23], Lemma 3.6 (see also Diaconis [24], Chapter 4, Lemma 5).

Lemma 12.1

Suppose that M is a countable, ergodic Markov chain with transition probabilities $P(\cdot, \cdot)$ and let $((X_t, Y_t), t \in \mathbb{N})$ be a coupling of M. Suppose further that $t : (0, 1] \to \mathbb{N}$ is a function such that $\Pr(X_{t(\epsilon)} \neq Y_{t(\epsilon)}) \leq \epsilon, \forall \epsilon \in (0, 1]$, uniformly over the choice of initial state (X_0, Y_0). Then the mixing time $\tau(\epsilon)$ of M is bounded above by $t(\epsilon)$. ◇

The transition $(X_t, Y_t) \to (X_{t+1}, Y_{t+1})$ in the coupling is defined by the following experiment:

(1) Select $v \in V$ u.a.r.
(2) Compute a permutation $g(G, X_t, Y_t)$ of C according to a procedure to be explained below.
(3) Choose a color $c \in C$ u.a.r.

(4) In the radiocoloring X_t (respectively Y_t) recolor vertex v with color c (respectively $g(c)$) to get a new radiocoloring X' (respectively Y').
(5) If X' (respectively Y') is a (valid) radiocoloring then $X_{t+1} = X'$ (respectively $Y_{t+1} = Y'$), else let $X_{t+1} = X_t$ (respectively $Y_{t+1} = Y_t$).

Note that whatever procedure is used to select the permutation g, the distribution of $g(c)$ is *uniform*, thus (X_t) and (Y_t) are both faithful copies of M.

We now remark that any set of vertices $F \subseteq V$ can have the same color in the graph G^2 only if they can have the same color in some radiocoloring of G. Thus, given a proper coloring of G^2 with λ' colors, we can construct a proper radiocoloring of G by giving the values (new colors) $1, 3, \ldots, 2\lambda' - 1$ in the color classes of G^2. Note that this transformation preserves the number of colors (but not the span).

Now let $A = A_t \subseteq V$ be the set of vertices on which the colorings of G^2 implied by X_t, Y_t agree and $Dim = D_t \subseteq V$ be the set on which they disagree. Let $d'(v)$ be the number of edges incident at v in G^2 that have one point in A and one in Dim. Clearly, if m' is the number of edges of G^2 spanning A, D, we get $\sum_{v\in A} d'(v) = \sum_{v\in D} d'(v) = m'$.

The procedure to compute $g(G, X_t, Y_t)$ is as follows:

(a) If $v \in D$ then g is the identity.
(b) If $v \in A$ then proceed as follows: Denote by N the set of neighbors of v in G^2. Define $C_x \subseteq C$ to be the set of all colors c, such that some vertex in N receives c in radiocoloring Y_t but no vertex in N receives c in radiocoloring Y_t. Let C_y be defined as C_x with the roles of X_t, Y_t interchanged. Observe $C_x \cap C_y = \emptyset$ and $|C_x|, |C_y| \leq d'(v)$. Let, w.l.o.g., $|C_x| \leq |C_y|$. Choose any subset $C'_y \subseteq C_y$ with $|C'_y| \leq |C_x|$ and let $C_x = \{c_1, \ldots, c_r\}$, $C'_y = \{c'_1, \ldots, c'_r\}$ be enumerations of C_x, $C_{y'}$ coming from the orderings of X_t, Y_t. Finally, let g be the permutation $(c_1, c'_1), \ldots, (c_r, c'_r)$, which interchanges the color sets C_x, $C_{y'}$ and leaves all other colors fixed.

It is clear that $|D_{t+1}| - |D_t| \in \{-1, 0, 1\}$.

(i) Consider first the probability that $|D_{t+1}| = |D_t| + 1$. For this event to occur, the vertex v selected in step (1) of the procedure for g must lie in A and hence we follow (b). If the new radiocolorings are to disagree at vertex v then the color c selected in line (3) must be an element of C_y. But $|C_y| \leq d'(v)$ hence

$$\Pr\{|D_{t+1}| = |D_t| + 1\} \leq \frac{1}{n}\sum_{v\in A}\frac{d'(v)}{\lambda} = \frac{m'}{\lambda \cdot n} \tag{12.6}$$

(ii) Now consider the probability that $|D_{t+1}| = |D_t| - 1$. For this to occur, the vertex v must lie in Dim and hence the permutation g selected in line (2) is the identity. For X_{t+1}, Y_{t+1} to agree at v, it is enough that color c selected in step (3) is different from all the colors that X_t, Y_t imply for the neighbors of v in G^2. The number of colors c that satisfy this is (by our previous results) at least $\lambda - 2(2\Delta + 25) + d'(v)$ hence,

$$\begin{aligned}\Pr\{|D_{t+1}| = |D_t| - 1\} &\geq \frac{1}{n}\sum_{v\in D}\frac{\lambda - 2(2\Delta + 25) + d'(v)}{\lambda}\\ &\geq \frac{\lambda - 2(2\Delta + 25)}{\lambda n}|D| + \frac{m'}{\lambda n}\end{aligned} \tag{12.7}$$

Define now $\alpha = \frac{\lambda - 2(2\Delta+25)}{\lambda n}$ and $\beta = \frac{m'}{\lambda n}$. So

$$\Pr\{|D_{t+1}| = |D_t| + 1\} \leq \beta$$

and $\Pr\{|D_{t+1}| = |D_t| - 1\} \geq \alpha|D_t| + \beta$

Given $\alpha > 0$, i.e. $\lambda > 2(2\Delta + 25)$, from Eq. (12.6) and Eq. (12.7), we get

$$\begin{aligned}E(|D_{t+1}|) &\leq \beta(|D_t| + 1) + (\alpha|D_t| + \beta)(|D_t| - 1) + (1 - \alpha|D_t| - 2\beta)|D_t|\\ &= (1 - \alpha)|D_t|\end{aligned}$$

Thus, from Bayes, we get $E(|D_{t+1}|) \leq (1-\alpha)^t |D_0| \leq n(1-\alpha)^t$
and since $|D_t|$ is a nonnegative random variable, we get, by the Markov inequality, that

$$\Pr\{D_t \neq 0\} \leq n(1-\alpha)^t \leq ne^{-\alpha t}$$

So, we note that, $\forall\, \epsilon > 0, \Pr\{D_t \neq \emptyset\} \leq \epsilon$ provided that $t \geq \frac{1}{\alpha} \ln\left(\frac{n}{\epsilon}\right)$ thus proving Theorem 12.2.

Theorem 12.2

Let G be a planar graph of maximum degree Δ on n vertices. Assuming $\lambda \geq 2(2\Delta + 25)$ the convergence time $\tau(\epsilon)$ of the Markov chain $M(G, \lambda)$ is bounded above by

$$\tau_x(\epsilon) \leq \frac{\lambda}{\lambda - 2(2\Delta + 25)} \;\; n \ln\left(\frac{n}{\epsilon}\right)$$

regardless of the initial state x. □

12.3.5 An FPRAS for Radiocolorings with λ Colors

We first provide the following definition.

Definition 12.5

A randomized approximation scheme for radiocolorings with λ colors of a planar graph G is a probabilistic algorithm that takes as input the graph G and an error bound $\epsilon > 0$ and outputs a number Y (a random variable) such that

$$\Pr\{(1-\epsilon)\, |R_\lambda(G)| \leq Y \leq (1+\epsilon)|R_\lambda(G)|\} \geq \frac{3}{4}$$

Such a scheme is said to be fully polynomial *if it runs in time polynomial in n and ϵ^{-1}. We abbreviate such schemes to* FPRAS.

The technique we employ is as in Ref. [18] and is fairly standard in the area. By using it we get the following theorem.

Theorem 12.3

There is an FPRAS for the number of radiocolorings of a planar graph G with λ colors, provided that $\lambda > 2(2\Delta + 25)$, *where Δ is the maximum degree of G.*

Proof

Recall that $R_\lambda(G)$ is the set of all radiocolorings of G with λ colors. Let m be the number of edges in G and let

$$G = G_m \supseteq G_{m-1} \supseteq \cdots \supseteq G_1 \supseteq G_0$$

be any sequence of graphs where G_{i-1} is obtained by G_i by removing a single edge. We can always erase an edge whose one node is of degree at most 5 in G_i. Clearly

$$|R_\lambda(G)| = \frac{|R_\lambda(G_m)|}{|R_\lambda(G_{m-1})|} \cdot \frac{|R_\lambda(G_{m-1})|}{|R_\lambda(G_{m-2})|} \cdots \frac{|R_\lambda(G_1)|}{|R_\lambda(G_0)|} \cdot |R_\lambda(G_0)|$$

But $|R_\lambda(G_0)| = \lambda^n$ for all kinds of colorings. The standard strategy is to estimate the ratio

$$\rho_i = \frac{|R_\lambda(G_i)|}{|R_\lambda(G_{i-1})|}$$

for each i, $1 \leq i \leq m$.

Suppose that graphs G_i, G_{i-1} differ in the edge $\{u, v\}$, which is present in G_i but not in G_{i-1}. Clearly, $R_\lambda(G_i) \subseteq R_\lambda(G_{i-1})$. Any radiocoloring in $R_\lambda(G_{i-1}) \setminus R_\lambda(G_i)$ assigns either the same color to u, v or the color values of u, v differ by only 1. Let $deg(v) \leq 5$ in G_i. So, we now have to recolor u with one of at least $\lambda - (2\Delta + 25)$, i.e., at least $2\Delta + 25$, colors (from Section 5 of Ref. [18]). Each radiocoloring of

$R_\lambda(G_i)$ can be obtained in at most one way by our algorithm of the previous section as the result of such a perturbation, thus

$$\frac{1}{2} \leq \frac{2\Delta + 25}{2(\Delta + 1) + 25} \leq \rho_i < 1 \tag{12.8}$$

To avoid trivialities, assume $0 < \epsilon \leq 1$, $n \geq 3$ and $\Delta > 2$.

Let $Z_i \in \{0, 1\}$ be the random variable obtained by simulating the Markov chain $M(G_{i-1}, \lambda)$ from any certain fixed initial state for

$$T = \frac{\lambda}{\lambda - 2(2\Delta + 25)} n \ln\left(\frac{4nm}{\epsilon}\right)$$

steps and returning to 1 if the final state is a member of $R_\lambda(G_i)$ and 0 else. Let $\mu_i = E(Z_i)$. By our theorem of rapid mixing, we have

$$\rho_i - \frac{\epsilon}{4m} \leq \mu_i \leq \rho_i + \frac{\epsilon}{4m}$$

and by Eq. (12.8), we get

$$\left(1 - \frac{\epsilon}{2m}\right)\rho_i \leq \mu_i \leq \left(1 + \frac{\epsilon}{2m}\right)\rho_i$$

As our estimator for $|R_\lambda(G)|$ we use

$$Y = \lambda^n Z_1 Z_2 \cdots Z_m$$

Note that $E(Y) = \lambda^n \mu_1 \mu_2 \cdots \mu_m$. But

$$Var(Y) \leq \frac{Var(Z_1 Z_2 \cdots Z_m)}{(\mu_1 \mu_2 \cdots \mu_m)^2} = \prod_{i=1}^{m}\left(1 + \frac{Var(Z_i)}{\mu_i^2}\right) - 1$$

By using standard techniques (as in Ref. [18]) one can easily show that Y satisfies the requirements for an FPRAS for the number of radiocolorings of graph G with λ colors $|R_\lambda(G)|$. □

References

[1] Vazirani, V. V., *Approximation Algorithms,* Springer, Berlin, 2001.

[2] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman and Co., New York, 1979.

[3] Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity,* Englewood Cliffs, NJ, 1982.

[4] Srinivason, A., Improved approximations of packing and covering problems, *Proc. STOC,* 2005.

[5] Hromkovic, J., *Design and Analysis of Randomized Algorithms,* EATCS Series, Springer, Berlin, 2005.

[6] Goemans, M. X. and Williamson, D. P., Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *JACM,* 42, 1115, 1995.

[7] Jerrum, M., Mathematical foundations of the Markov chain Monte Carlo method, in *Probabilistic Methods for Algorithmic Discrete Mathematics,* Habib, M., McDiarmid, C., Ramirez-Alfonsin, J., and Reed, B., Eds., Springer Verlag, 1998, p. 116.

[8] Knuth, D. E., Estimating the efficiency of backtrack problems, *Math. Comput.,* 29, 121, 1975.

[9] Rusmussen, L. E., Approximating the permanent: a simple approach, *Random Struct. Algorithms,* 5, 349, 1994.

[10] Jerrum, M. and Sinclair, A., The Markov chain Monte Carlo method: an approach to approximate counting and integration, in *Approximation Algorithms for NP-Hard Problems,* PWS Publishing Co., Boston, MA, 1997, p. 482.

[11] Hale, W. K., Frequency assignment: theory and applications, *Proc. of the IEEE,* 68(12), 1980, 1497.

[12] Griggs, J. and Liu, D., Minimum span channel assignments, in *Recent Advances in Radio Channel Assignments, in the Ninth SIAM Conference on Discrete Mathematics*, Toronto, Canada, 1998.

[13] Fotakis, D., Pantziou, G., Pentaris, G., and Spirakis, P., Frequency assignment in mobile and radio networks, *Networks in Distributed Computing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science,* AMS, Vol. 45, New Jersey, 1999, p. 73.

[14] Katsela, I. and Nagshineh, M., Channel assignment schemes for cellular mobile telecommunication systems, *IEEE Personal Commun. Complexity,* 1070, 115–134, 1996.

[15] Bertossi, A. A. and Bonuccelli, M. A., Code assignment for hidden terminal interference avoidance in multihop packet radio networks, *IEEE/ACM Trans. Networking,* 3(4), 441, 1995.

[16] Ramanathan, S. and Loyd, E. R., The complexity of distance 2-coloring, *Proc. of the 4th Int. Conf. of Computing and Information,* 1992, p. 71.

[17] Fotakis, D., Nikoletseas, S., Papadopoulou, V., and Spirakis, P., Radiocolorings in planar graphs: complexity and approximations, *Theor. Comput. Sci. J.,* 340(3), 205, 2005. Also, in *Proc. of the 25th Int. Symp. on Mathematical Foundations of Computer Science (MFCS 2000),* Lecture Notes in Computer Science, Vol. 1893, Springer, Berlin, 2000, p. 363.

[18] Jerrum, M., A very simple algorithm for estimating the number of k-colourings of a low degree graph, *Random Struct. Algorithms,* 7, 157, 1994.

[19] Andreou, M., Fotakis, D., Nikoletseas, S., Papadopoulou, V., and Spirakis, P., On radiocoloring hierarchically specified planar graphs: PSPACE-completeness and approximations, *Proc. of the 27th Int. Symp. on Mathematical Foundations of Computer Science (MFCS),* Lecture Notes in Computer Science, Vol. 2420, Springer, Berlin, 2002, p. 81.

[20] Fotakis, D., Nikoletseas, S., Papadopoulou, V., and Spirakis, P., Radiocolorings in periodic planar graphs: PSPACE-completeness and efficient approximations for the optimal range of frequencies, *Proc. of the 28th Int. Workshop on Graph-Theoretic Concepts in Computer Science,* Lecture Notes in Computer Science, Vol. 2573, Springer, Berlin, 2002, p. 223.

[21] Fotakis, D., Nikoletseas, S., Papadopoulou, V., and Spirakis, P., Hardness results and efficient approximations for frequency assignment problems: radio labelling and radio coloring, *J. Comput. Artif. Intell. (CAI),* 20(2), 121, 2001.

[22] Habib, M., McDiarmid, C., Ramirez-Alfonsin, J., and Reed, B., Eds., *Probabilistic Methods for Algorithmic Discrete Mathematics*, Springer, Berlin, 1998.

[23] Aldous, D., Random walks in finite groups and rapidly mixing Markov chains, in *Seminaire de Probabilites XVII 1981/82, Springer Lecture Notes in Mathematics*, Dold, A. and Eckmann, B., Eds. Vol. 986, Springer, Berlin, 1982, p. 243.

[24] Diaconis, P., Group Representations in Probability and Statistics, Institute of Mathematical Statistics, Hayward, CA, 1988.

13

Distributed Approximation Algorithms via LP-Duality and Randomization

Devdatt Dubhashi
Chalmers University

Fabrizio Grandoni
University of Rome "La Sapienza"

Alessandro Panconesi
University of Rome "La Sapienza"

13.1 Introduction

The spread of computer networks, from sensor networks to the Internet, creates an ever-growing need for efficient *distributed* algorithms. In such scenarios, familiar combinatorial structures such as spanning trees and dominating sets are often useful for a variety of tasks. Others, like maximal independent sets, turn out to be a very useful primitive for computing other structures. In a distributed setting, where transmission of messages can be orders of magnitude slower than local computation, the expensive resource is communication. Therefore, the running time of an algorithm is given by the number of communication rounds that are needed by the algorithm. This will be made precise below.

In what follows we will survey a few problems and their solutions in a distributed setting: dominating sets, edge and vertex colorings, matchings, vertex covers, and minimum spanning trees. These problems were chosen for a variety of reasons: they are fundamental combinatorial structures; computing them is useful in distributed settings; and they serve to illustrate some interesting techniques and methods.

Randomization, whose virtues are well known to people coping with parallel and distributed algorithms, will be a recurrent theme. In fact, only rarely it has been possible to develop deterministic distributed algorithms for nontrivial combinatorial optimization problems. Here, in the section on vertex covers, we will discuss a novel and promising approach based on the primal-dual methodology to develop efficient, distributed deterministic algorithms. One of the main uses of randomization in distributed scenarios is to break the symmetry. This is well illustrated in Section 13.2. discussing dominating sets. Often, the analysis of simple randomized protocols requires deep results from probability theory. This will be illustrated in Section 13.3, where martingale methods are used to analyze some simple, and yet almost optimal, distributed algorithms for edge coloring. The area of distributed algorithms for graph problems is perhaps unique in complexity theory because it is possible to derive several nontrivial absolute lower bounds (that is, not relying on special complexity assumptions such as $\mathsf{P} \neq \mathsf{NP}$). This will be discussed in Section 13.6.

Let us then define the computation model. We have a message-passing, synchronous network: vertices are processors, edges are communication links, and the network is synchronous. Communication proceeds in synchronous *rounds*: in each round, every vertex sends messages to its neighbors, receives messages from its neighbors, and does some amount of local computation. It is also assumed that each vertex has a unique identifier. In the case of randomized algorithms each node of the network has access to its own source of random bits. *In this model, the running time is the number of communication rounds.* This will be our notion of "time." As remarked, this is a very reasonable first approximation since typically sending a message is orders of magnitude slower than performing local computation.

Although we place no limits on the amount of local computation, the algorithms we describe perform polynomial-time local computations only. Under the assumption that local computations are polynomial time several of the algorithms that we describe are "state of the art," in the sense that their approximation guarantee is the same, or comparable, to that obtainable in a centralized setting. It is remarkable that this can be achieved in a distributed setting.

The model is in some sense orthogonal to the Parallel Random Access Machine (PRAM) model for parallel computation where a set of polynomially many, synchronous processors access a shared memory. There communication is free: any two processors can communicate in constant time via the shared memory. In the distributed model, in contrast, messages are routed through the network and therefore the cost of sending a message is at least proportional to the length of the shortest path between the two nodes. On the other hand, local computation is inexpensive, while this is the expensive resource in the PRAM model.

Note that there is a trivial universal algorithm that always works: The network elects a leader which then collects the entire topology of the network, computes the answers, and notifies them to the other nodes. This will take a time proportional to the diameter of the network, which can be as large as n, the number of nodes. In general, we will be looking for algorithms that take polylogarithmically, in n, many communication rounds, regardless of the diameter of the network. Such algorithms will be called *efficient.*

Note the challenge here: if a protocol runs for t rounds then each processor can receive messages from nodes at distance at most t. For small values of t this means that the network is computing a global function of itself by relying on local information alone.

13.2 Small Dominating Sets

In this section we study the minimum dominating set (MDS) problem. The advent of wireless networks gives a new significance to the problem since (connected) dominating sets are the structure of choice to set up the routing infrastructure of such ad hoc networks, the so-called backbone (see, for instance, Ref. [1] and references therein). In the sequel we describe a nice algorithm from Ref. [2] for computing small dominating sets. The algorithm is in essence an elegant parallelization of the well-known greedy heuristic for set cover [3,4]. Randomness is a key ingredient in the parallelization. The algorithm computes, on any input graph, a dominating set of size at most $O(\log \Delta)opt$, where as customary Δ denotes the maximum degree of the graph and opt is the smallest size of a dominating set in the input graph. By "computing a dominating set" we mean that at the end of the protocol every vertex decides whether it is in the dominating set or not. The algorithm was originally developed for the PRAM model but, as we will show, it can be implemented distributively. It is noteworthy that the approximation bound is essentially the "best possible" under the assumption that every node performs a polynomially bounded computation during every round. "Best possible" means that there exists a constant $c > 0$ such that a $c \log n$-approximation would imply that $\mathsf{P} = \mathsf{NP}$ [5], while a $(c \ln n)$-approximation, for a constant $c < 1$, would imply that NP could be solved exactly by means of slightly superpolynomial algorithms [6,7].

We shall then describe a surprisingly simple deterministic algorithm that, building on top of the dominating set algorithm, computes a "best possible" connected dominating set, in $O(\log n)$ additional communication rounds [8].

There are other nice algorithms to compute dominating sets efficiently in a distributed setting. The algorithm in Ref. [9] is a somewhat different parallelization of the greedy algorithm, while Ref. [10] explores an interesting trade-off between the number of rounds of the algorithm and the quality of the approximation that it achieves. This paper makes use of LP-based methods, an issue that we will explore in Section 13.5.

13.2.1 Greedy

Let us start by reviewing the well-known greedy heuristic for set cover. Greedy repeatedly picks the set of minimum unit cost, creating a new instance after every choice by removing the points just covered. More formally, let (X, F, c) be a set cover instance, where X is a ground set of elements and $F := \{S_i : S_i \subseteq X, i \in [m]\}$ is a family of nonempty subsets of X with positive costs $c(S) > 0$. The goal is to select a subfamily of minimum cost that covers the ground set. The cost of a subfamily is the sum of the costs of each set in the subfamily.

Dominating set is a special case of set cover. A graph G with positive weights $c(u)$, $u \in V(G)$, can be viewed as a set system $\{S_u : u \in V(G)\}$ with $S_u := N(u) \cup \{u\}$, where $N(u)$ is the set of neighbors of u, and $c(S_u) := c(u)$.

Given a set cover instance $I := (X, F, c)$, let $c(e) := \min_{e \in S \in F} \frac{c(S)}{|S|}$ be the *cost* of the element $e \in X$. This is the cheapest way to cover e where we do the accounting in the following natural way: when we pick a set, its cost is distributed equally to all elements it covers. An algorithm A may pick a certain set S' at this stage, then in this accounting scheme, each element $e \in S'$ pays the *price* $p(e) := \frac{c(S')}{|S'|}$. Once set s' is picked, we create a new instance I' with ground set $X' := X - \hat{S}$ and set system F' whose sets are defined as $S'_i := S_i - \hat{S}$. The new costs coincide with the old ones: $c(S') = c(S)$, for all $S \in F'$. The algorithm continues in the same fashion until all elements are covered.

Greedy selects a set $\hat{S}$ at each stage that realizes the minimum unit cost, i.e., $p(e) = c(e)$ at each stage. In other words, greedy repeatedly selects the set that guarantees the smallest unit price. For the discussion to follow concerning the distributed version of the algorithm it is important to notice that each element e is assigned a price tag $p(e)$ only once, at the time when it is covered by greedy. For a subset $A \subseteq X$, let $g(A) := \sum_{e \in A} p(e)$. Then $g(X)$, the sum of the unit prices, is the total cost incurred by greedy. The crux of the analysis is the next lemma.

Lemma 13.1

For any set S, $g(S) \leq H_{|S|} c(S)$ where $H_k := 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k}$ is the kth harmonic number.

Proof

Sort the elements of S according to the time when they are covered by greedy, breaking ties arbitrarily. Let $e_1, e_2, \cdots, e_k$ be this numbering. When greedy covers e_i it must be that $p(e_i) \leq \frac{c(S)}{k-i}$. The claim follows. □

Clearly we have that $g(A \cup B) \leq g(A) + g(B)$. Denoting with C^* an optimal cover, we have, by Lemma 13.1,

$$g(X) = g\left(\cup_{S \in C^*} S\right) \leq \sum_{S \in C^*} g(S) \leq \sum_{S \in C^*} H_{|S|} c(S) \leq \max_S H_{|S|} \sum_{S \in C^*} c(S) \leq \max_S H_{|S|} opt$$

It is well known that $\log k \leq H_k \leq \log k + 1$. In the case of the dominating set the bound becomes

$$g(X) \leq H_{\Delta+1} opt = O(\log \Delta) opt$$

where Δ is the maximum degree of the graph.

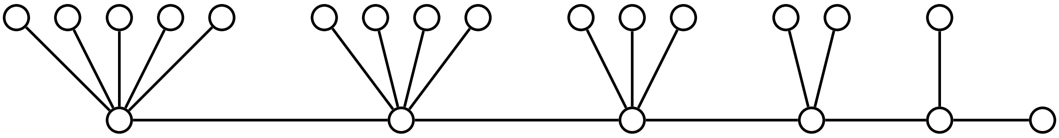


FIGURE 13.1 Example of lower-bound graph for $k = 6$. The number of nodes is $n = k(k+1)/2 = \Theta(k^2)$. The bottom nodes are selected by greedy, one by one from left to right. The number of rounds is $k - 1$.

13.2.2 Greedy Hordes

We now proceed to parallelize greedy. Figure 13.1 shows that the number of steps taken by greedy can be $\Omega(\sqrt{n})$. The problem lies in the fact that at any stage there is just one candidate set that gives the minimum unit cost $\hat{c}$. It is to get around this problem that we introduce the following notion. A *candidate* is any set S such that

$$\hat{c} \leq \frac{c(S)}{|S|} \leq 2\hat{c} \tag{13.1}$$

Let us modify greedy in such a way that, at any step, it selects any set satisfying this condition. With this modification, the solution computed by greedy will still be at most $O(\log n)opt$ since the algorithm pays at most twice the smallest unit price the overall we lose only a factor of 2 in the approximation.

Suppose now that the algorithm is modified in such a way that it adds to the solution all candidates satisfying Eq. (13.1). With this modification, the graphs of Figure 13.1 will be covered in $O(\log n)$ steps. But as the example of the clique shows (all the nodes are selected) this increase in speed destroys the approximation guarantee. This is because the key requirement of the sequential greedy procedure is violated. In the sequential procedure, the price $p(e)$ is paid only once, at the time when e is covered. If we do things in parallel we need to keep two conflicting requirements in mind: picking too many sets at once can destroy the approximation guarantee, but picking too few can result in slow progress. And we must come up with a charging scheme to distribute the costs among the elements in a manner similar to the sequential case.

Rajagopalan and Vazirani solved this problem by devising a scheme that picks enough sets to make progress but at the same time retains the parsimonius accounting of costs like in the sequential version. Specifically, for every set S selected by greedy, the cost $c(S)$ will be distributed among the elements of a subset $T \subset S$ of at least $|S|/4$ elements. Crucially, the elements of T will be charged only once. If we can do this then we will lose another factor of 4 in the approximation guarantee with respect to greedy, all in all losing a factor of 8.

The scheme works as follows: line up the candidate sets satisfying Eq. (13.1) on one side and all the elements on the other. The elements are thought of as *voters* and cast their vote for one of the candidate sets containing them by an *election*. An election is conducted as follows:

- A random permutation of the candidates is computed.
- Among all the candidate sets that contain it, each voter votes for that set which has the lowest number in the permutation.
- A candidate is elected if it obtains at least $\frac{1}{4}$ of the votes of its electorate. Elected candidates enter the set cover being constructed.

The cost of the set can now be distributed equally among the elements that voted for it, i.e., at least a quarter of the elements.

Let us now describe the distributed implementation of this scheme in the specific case of the set system corresponding to the dominating set problem. During the execution nodes can be in four different states:

- They can be *free*. Initially all vertices are free.
- They can be *dominated*.

- They can be *dominators.* Dominators are added to the dominating set and removed from the graph.
- They can be *out.* Vertices are out when they are dominated and have no free neighbors. These vertices are removed from the graph since they can play no useful role.

The algorithm is a sequence of $\log \Delta$ *phases* during which the following invariant is maintained, with high probability. At the beginning of phase i, $i = 1, 2, \ldots, \log \Delta$, the maximum degree of the graph is at most $\Delta/2^{i-1}$. The candidates during phase i are all those vertices whose degree is in the interval $(\Delta/2^i, \Delta/2^{i-1}]$ i.e., they satisfy condition (13.1). Note that candidates can be free or dominated vertices. The voters are those free nodes that are adjacent to a candidate. This naturally defines a bipartite graph with candidates on one side, voters on the other, and edges that represent domination relationships. Each phase consists of a series of $O(\log n)$ elections. A free vertex can appear on both sides, since a free vertex can dominate itself. We shall refer to the neighbors of a candidate c in the bipartite graph as the *electorate* of c, and to the neighbors of a voter v as the *pool* of v. Elections are carried out and elected candidates enter the dominating set.

Step 1 of each election seems to require global synchronization, but a random permutation can be generated if the value of n is known. If each element picks a random number between 1 and n^k then with probability $1 - 1/n^{k-1}$ all choices will be distinct. Thus, the probability that there is a collision is negligible during the entire execution of the algorithm.

After every election, nodes are removed for two different reasons. Elected nodes disappear from the candidate side of the bipartition, while their neighbors disappear from the other side since they are no more free. In the analysis we will show that after one election the expected number of edges that disappear from the bipartite graph is a constant fraction of the total. This automatically implies that the total number of elections to remove all edges from the graph is $O(\log n)$ with overwhelming probability. More precisely, for any $c > 0$ there is $\alpha > 0$ such that, the probability that the bipartite graph is nonempty after $\alpha \log n$ elections is at most n^{-c} [11,12]. It follows that α can be chosen in such a way that the probability that some phase does not end successfully is negligible.

A voter v is *influential* for a candidate c if at least $\frac{3}{4}$ of the voters in c's electorate have degree no greater than that of v. Let $d(v)$ denote the degree of v.

Lemma 13.2

For any two voters v and w, $d(v) \geq d(w)$, in c's electorate, $\mathsf{Pr}[w \textit{ votes } c \mid v \textit{ votes } c] \geq \frac{1}{2}$.

Proof

Let N_b denote the number of neighbors that v and w have in common, let N_v the number of neighbors of v that are not neighbors of w, and N_w the number of neighbors of w that are not neighbors of v. Then,

$$\mathsf{Pr}[w \text{ votes } c \mid v \text{ votes } c] = \frac{\mathsf{Pr}[w \text{ votes } c, v \text{ votes } c]}{\mathsf{Pr}[v \text{ votes } c]} = \frac{N_v + N_b}{N_v + N_b + N_w} \geq \frac{1}{2} \qquad \square$$

Lemma 13.3

Let v be an influential voter for c. Then, $\mathsf{Pr}[c \textit{ is elected} \mid v \textit{ votes } c] \geq \frac{1}{6}$.

Proof

Let $X :=$ (# votes for c) and $Y := c - X$ where, with abuse of notation we use c to denote the size of c's electorate. Then, by Lemma 13.2

$$\mathsf{E}[X \mid v \text{ votes } c] \geq \sum_{w:\, d(w) \leq d(v)} \mathsf{Pr}[w \text{ votes } c \mid v \text{ votes } c] \geq \frac{3}{8}c$$

Applying Markov's inequality to Y we get

$$\begin{aligned}\mathsf{Pr}[c \text{ not elected} \mid v \text{ votes } c] &= \mathsf{Pr}[X < c/4 \mid v \text{ votes } c] = \mathsf{Pr}[Y \geq 3c/4 \mid v \text{ votes } c] \\ &\leq \frac{4\,\mathsf{E}[Y \mid v \text{ votes } c]}{3c} = \frac{4(c - \mathsf{E}[X \mid v \text{ votes } c])}{3c} \leq \frac{5}{6}\end{aligned}$$

The claim follows. □

Lemma 13.4

Fix a phase and let m denote the total number of edges in the bipartite graph at any stage in this phase. Let X denote the number of edges removed from the bipartite graph after one election. Then, $\mathsf{E}[X] \geq \frac{m}{24}$.

Proof

An edge vc is *good* if v is influential for c. By definition, at least $\frac{1}{4}$ of the edges are good. Then,

$$\begin{aligned}
\mathsf{E}[X] &= \sum_{vc} \mathsf{Pr}[c \text{ is elected, } v \text{ votes } c]d(v) \\
&\geq \sum_{vc \text{ good}} \mathsf{Pr}[c \text{ is elected, } v \text{ votes } c]d(v) \\
&\geq \sum_{vc \text{ good}} \mathsf{Pr}[v \text{ votes } c]\ \mathsf{Pr}[c \text{ is elected} \mid v \text{ votes } c]d(v) \\
&= \sum_{vc \text{ good}} \mathsf{Pr}[c \text{ is elected} \mid v \text{ votes } c] \\
&\geq \frac{m}{24} \quad \text{by Lemma 13.3}
\end{aligned}$$

□

As remarked, this lemma implies that, with high probability, $O(\log n)$ rounds are sufficient for every phase. The resulting running time is $O(\log n \log \Delta)$ communication rounds, while the approximation guarantee is $O(\log \Delta)$. Vertices must know n to compute a permutation and to run the correct number of elections, and they must know Δ to decide whether they are candidates at the current phase. Alternatively, if only the value of n is known, the algorithm can execute $O(\log n)$ phases, for a total of $O(\log^2 n)$ many rounds.

13.2.3 Small Connected Dominating Sets

In this section we develop an efficient distributed algorithm for computing "best possible" connected dominating sets. Again, by this we mean that the protocol computes a connected dominating set of size at most $O(\log \Delta)$ times the optimum. Nowadays, connected dominating sets are quite relevant from the application point of view since they are the solution of choice for setting up the backbones of self-organizing networks such as ad hoc and sensor networks (see Ref. [1] and references therein). A backbone is a subnetwork that is in charge of administering the traffic inside a network.

What is remarkable from the algorithmic point of view is that connectivity is a strong global property, and yet we will be able to obtain it by means of a distributed algorithm that relies on local information alone. The overall strategy can be summarized as follows:

- Compute a small dominating set.
- Connect it up using a sparse spanning network.

We saw in the previous section how to take care of step 1. To connect a dominating set we can proceed as follows. Let D be the dominating set in the graph G created after step 1. Consider an auxiliary graph H with vertex set D and where any two $u, v \in D$ that are at distance 1, 2, or 3 in G are connected by an edge in H. It is easy to see that H is connected if G is (which we assume). Every edge in H corresponds to a path with 0, 1, or 2 vertices in G. If we inserted all such vertices we would still have a dominating set, since adding vertices can only improve domination. The resulting set would however be too large in general, since H can have as many as $|D|^2$ edges, each contributing with two vertices. The best way to connect D up would be to compute a spanning tree T. If we could do this, adding to D all vertices lying on paths corresponding to the edges of T, we would obtain the desired approximation since $E(T) = |D| - 1$ and recalling that $|D|$ is a $O(\log \Delta)$-approximation. Therefore, denoting with D^* and C^* an optimal dominating and connected dominating set, respectively, we would have (with some abuse of notation) that $|D \cup V(T)| \leq 3|D| \leq O(\log \Delta)|D^*| \leq O(\log \Delta)|C^*|$.

The problem however is that, as we discuss in Section 13.6.1, computing a spanning tree takes time $\Omega(\sqrt{n})$. In what follows we show a very simple algorithm that computes, in $O(\log |V(G)|)$ many communication rounds, a network $S \subset H$ such that (a) S is connected, (b) $|E(S)| = O(|D|)$, and (c) $V(S) = D$. In words, S is a sparse connected network that spans the whole of D with linearly many edges. If we can compute such an S than we will have a connected dominating set of size at most $O(\log \Delta)$ times the optimum. S will not be acyclic but this is actually a positive thing since it makes S more resilient to failures. In fault-prone environments such as ad hoc and sensor networks this kind of redundancy is actually very useful. The key to computing S is given by the following lemma (see, for instance Ref. [13, Lemma 15.3.1]). Recall that the *girth* of a graph G is the length of the shortest cycle in G.

Lemma 13.5

Let $G = (V, E)$ be a graph of girth g, and let $m := |E|$ and $n := |V|$. Then, $m \leq n + n^{1+2/(g-1)}$.

Proof

Assume $g = 2k + 1$ and let $d := \frac{m}{n}$. Consider the following procedure. As long as there is a vertex whose degree is less than d, remove it. Every time we remove a vertex the new minimum degree is at least as large as the old one. Therefore, this procedure ends with a graph whose minimum degree is at least d. Now pick any vertex in this graph and start a breadth first search. This generates a tree in which the root has at least d children and every other node has at least $d - 1$ children. Moreover, assigning level 0 to the root, this tree is a real tree up to and including level $k - 1$, i.e., no two vertices of this Breadth-First Search (BFS) exploration coincide up to that level. Therefore,

$$n \geq 1 + d + d(d-1) + \cdots + d(d-1)^{k-1} \geq (d-1)^k$$

Recalling the definition of d, the claim follows. The proof for the case $g = 2k$ is analogous. □

Note that if $g = 2\log n + 1$ then $m \leq 3n$. Define a cycle to be *small* if it is of length at most $2\log n + 1$. The following amazingly simple protocol removes all small cycles while, crucially, preserving connectivity:

- If an edge is the smallest in some small cycle, it is deleted.

Assume that every edge in the graph has a unique identifier. An edge is smaller than another edge if its identifier is smaller than that of the other edge. It is clear that every small cycle is destroyed. The next lemma shows that connectivity is preserved.

Lemma 13.6

The above protocol preserves connectivity.

Proof

Sort the edges by increasing IDs and consider the following sequential procedure. At the beginning all edges are present in the graph. At step i edge e_i is considered. If e_i is in a small cycle then it is removed. This breaks all small cycles and preserves connectivity, since an edge is removed only when there is another path connecting its endpoints. The claim follows by observing that the sequential procedure and the distributed protocol remove the same set of edges. □

To implement the protocol we only need to determine the small cycles to which an edge belongs. This can be done by a BFS of depth $O(\log n)$ starting from every vertex. If edges do not have distinct IDs to start with they can be generated by selecting a random number in the range $[m^3]$, which ensures that all IDs are distinct with overwhelming probability. This requires the value of n or m to be known. This sparsification technique appears to be quite effective in practice [1].

13.3 Coloring: The Extraordinary Career of a Trivial Algorithm

Consider the following sequential greedy algorithm to color the vertices of an input graph with $\Delta + 1$ colors, where Δ is the maximum degree: pick a vertex, give it a color not assigned to any of its neighbors; repeat until all vertices are colored. In general, Δ can be quite far from the optimal value $\chi(G)$ but it should not be forgotten that the chromatic number is one of the most difficult combinatorial problems to approximate [14–16].

In this section we will see how efficient distributed implementations of this simple algorithm lead to surprisingly strong results for vertex and especially edge coloring. Consider first the following distributed implementation. Each vertex u is initially given a list of colors $L_u := \{1, 2, \ldots, \Delta + 1\}$. Computation proceeds in rounds, until the graph is colored. One round is as follows: each uncolored vertex u picks a tentative color $t_u \in L_u$; if no neighboring vertex has chosen the same tentative color, t_u becomes the final color of u, and u stops. Otherwise L_u is updated by removing from it all colors assigned to neighbors of u at the current round. We shall refer to this as the *trivial algorithm*. It is apparent that the algorithm is distributed.

The trivial algorithm is clearly correct. An elementary, but nontrivial analysis shows that the probability that an uncolored vertex colors itself in one round is at least $\frac{1}{4}$ [17]. As we discussed in the previous section, this implies that the algorithm will color the entire network within $O(\log n)$ communication rounds, with high probability.

The following slight generalization is easier to analyze. At the beginning of every round, uncolored vertices are *asleep* and wake up with probability p. The vertices that wake up execute the round exactly as described earlier. At the end of the round, uncolored vertices go back to sleep. In other words, the previous algorithm is obtained by setting $p = 1$. In the sequel we will refer to this generalization as the (generalized) trivial algorithm. Luby analyzed this algorithm for $p = \frac{1}{2}$ [18]. Heuristically, it is not hard to see why the algorithm makes progress in this case. Assume u is awake. The expected number of neighbors of u that wake up is $d(u)/2 \leq |L_u|/2$.

In the worst case, these neighbors will pick different colors and all these colors will be in L_u. Even then, u will have probability at least $\frac{1}{2}$ to pick a color that creates no conflict. Thus, with probability $\frac{1}{2}$ a vertex wakes up and, given this, with probability at least $\frac{1}{2}$ it colors itself. The next proposition formalizes this heuristic argument.

Proposition 13.1

When $p = \frac{1}{2}$ the probability that an uncolored vertex colors itself in one round is at least $\frac{1}{4}$.

Proof

Let t_u denote the tentative color choice of a vertex u.

$$\begin{aligned}
\Pr[u \text{ does not color} \mid u \text{ wakes up}] &= \Pr[\exists v \in N(u)\ t_u = t_v \mid u \text{ wakes up}] \\
&\leq \sum_{v \in N(u)} \Pr[t_u = t_v \mid u \text{ wakes up}] \\
&= \sum_{v \in N(u)} \Pr[t_u = t_v \mid u \text{ and } v \text{ wake up}] \Pr[v \text{ wakes up}] \\
&= \sum_{v \in N(u)} \frac{|L_u \cap L_v|}{|L_v||L_u|} \frac{1}{2} \leq \sum_{v \in N(u)} \frac{1}{|L_u|} \frac{1}{2} \leq \frac{1}{2}
\end{aligned}$$

Therefore,

$$\Pr[u \text{ colors itself}] = \Pr[u \text{ colors itself} \mid u \text{ wakes up}] \Pr[u \text{ wakes up}] \geq \frac{1}{4} \qquad \square$$

Note that the trivial algorithm works just as well if the lists are initialized as $L_u := \{1, 2, \ldots, d(u) + 1\}$, for all $u \in V(G)$, for any value of $p > 0$. Interestingly, in practice, with $p = 1$ the trivial algorithm is much faster than Luby's one. In fact, experimentally, the speed of the algorithm increases regularly and monotonically as p tends to 1 [19].

In the distributed model we can simulate the trivial algorithm for the line graph with constant-time overhead. In this case, the algorithm will be executed by the edges rather than the vertices, each edge e having its own list L_e. In this fashion we can compute edge colorings that are approximated by a factor of 2 (since $2\Delta - 1$ colors are used). It is a challenging open problem whether an $O(\Delta)$-approximation can be computed deterministically in the distributed model. The best known result so far is an $O(\Delta \log n)$-approximation [20]. But the real surprise is that the trivial algorithm computes near-optimal edge colorings!

Vizing's theorem shows that every graph G can be edge colored sequentially in polynomial time with Δ or $\Delta + 1$ colors (see, for instance, Ref. [21]). The proof is in fact a polynomial-time sequential algorithm for achieving a $\Delta + 1$ coloring. Thus edge coloring can be well approximated. It is a very challenging open problem whether colorings as good as these can be computed fast in a distributed model.

If the edge lists L_e's are initialized to contain just a bit more than Δ colors, say $|L_e| = (1 + \epsilon)\Delta$ for all e, then the trivial algorithm will edge color the graph within $O(\log n)$ communication rounds. Here ϵ can be any fixed, positive constant. Some lists can run out of colors and, consequently, the algorithm can fail, but this happens with a probability that goes to 0 as n, the number of vertices, grows. All this is true, provided that the minimum degree $\delta(G)$ is large enough, i.e., $\delta(G) \gg \log n$ [22,23]. For Δ-regular graphs the condition becomes $\Delta \gg \log n$.

In fact, the trivial algorithm has in store more surprises. If the input graph is Δ-regular and has no triangles, it colors the vertices of the graph using only $O(\Delta / \log \Delta)$ colors. This is in general optimal, since there are infinite families of triangle-free graphs that need these many colors [24]. Again, the algorithm fails with negligible probability, provided that $\Delta \gg \log n$. For the algorithm to work, the value of p must be set to a value that depends on the round: small initially, it grows quickly to 1 [25].

The condition $\Delta \gg \log n$ appears repeatedly. The reason is that these algorithms are based on powerful martingale inequalities and this condition is needed to make them work. These probabilistic inequalities are the subject of the next section.

13.3.1 Coloring with Martingales

Let $f(X_1, \ldots, X_n)$ be a function for which we can compute $\mathsf{E}[f]$, and let the X_i's be independent. Assume moreover that the following Lipshitz condition (with respect to the Hamming distance) holds:

$$|f(X) - f(Y)| \leq c_i \tag{13.2}$$

whenever $X := (x_1, \ldots, x_n)$ and $Y := (y_1, \ldots, y_n)$ differ only in the ith coordinate. Then, f is sharply concentrated around its mean:

$$\Pr[|f - \mathsf{E}[f]| > t] \leq 2e^{-2t^2/\Sigma_i c_i^2} \tag{13.3}$$

This is the simplest of a series of powerful concentration inequalities dubbed the method of bounded differences (MOBD) [26]. The method is based on martingale inequalities (we refer the reader to the thorough and quite accessible treatment in Ref. [12]). In words, if a function does not depend too much on any coordinate then it is almost constant.

To appreciate the power and ease of use of Eq. (13.3) we derive the well-known Chernoff–Hoeffding bound (see, among others Refs. [12,27,28]). This bound states that if $X := \sum_{i=1}^{n} X_i$ is the sum of independent, binary random variables $X_i \in \{0, 1\}$, then X is concentrated around its mean: $\Pr[|X - \mathsf{E}[X]| > t] \leq 2e^{-2t^2/n}$. This captures the well-known fact that if a fair coin is flipped many times we expect HEADS to occur roughly 50% of the time, and this bound gives precise probability estimates of deviating from the mean. This bound can be recovered from Eq. (13.3) simply by defining $f := X$ and by noticing that condition (13.2) holds with $c_i = 1$.

We now apply the MOBD to the analysis of the trivial algorithm in a simplified setting. Let us assume that the network is a triangle-free, d-regular graph. We analyze what happens to the degree of a node after the first round. The probability with which an edge colors itself is $\left(1 - \frac{1}{d}\right)^{2d-2} \sim \frac{1}{e^2}$. Therefore, denoting with f the new degree of vertex u, we have that $\mathsf{E}[f] = \Theta(d)$. At first blush it may seem that the value of f depends on the tentative color choices of $\Theta(d^2)$ edges: those incident on u and the edges incident on them. But it is possible to express f as a function of $2d$ variables only, as follows. For every $v \in N(u)$ consider the bundle of $d - 1$ edges incident on v that are not incident on u, and treat this bundle as a single random variable, denoted as B_v. B_v is a random vector with $d-1$ components, each specifying the tentative color choice of an edge incident on v (except uv). Furthermore, for every edge $e = uv$, let X_e denote e's color choice. Thus, f depends on d variables of type X_e and on d variables of type B_v. What is the effect of these variables on f? If we change the value of a fixed X_e, and keep all remaining variables the same, this color change can affect at most two edges (one of which is e itself). The resulting c_e is 2. The cumulative effect of the first d variables of type X_e is therefore $4d$.

Note now that since the network is triangle-free, changing the value of a bundle B_v can only affect the edge uv the bundle is incident to. Thus, the effect of changing B_v while keeping everything else fixed is 1. Summing up, we get a total effect of $\sum_i c_i^2 = 5d$. Plugging in this value in Eq. (13.3), for $t = \epsilon d$, where $1 > \epsilon > 0$ we get, $\Pr[|f - \mathsf{E}[f]| > \epsilon d] \leq 2e^{-2\epsilon^2 d/5}$. We can see here why it is important to have $d \gg \log n$. With this condition, the bound is strong enough to hold for all vertices and all rounds simultaneously. In fact, a value $d = \Theta(\log n)$ would seem to be enough, but the error terms accumulate as the algorithm progresses. To counter this cumulative effect, we must have $d \gg \log n$.

This establishes that the graph stays almost regular after one round (and in fact at all times), with high probability. For the full analysis one has to keep track of several other quantities besides vertex degrees, such as the size of the color lists. While the full analysis of the algorithm is beyond the scope of this survey, this simple example already clarifies some of the issues. For instance, if the graph is not triangle-free, then the effect of a bundle can be much greater than 1. To cope with this, more powerful inequalities, and a more sophisticated analysis, are needed [12,22,23,29]. We remark that in general these inequalities do not even require the variables X_i to be independent. In fact, only the following bounded difference condition is required:

$$|\mathsf{E}[f|X_1, \ldots, X_{i-1}, X_i = a] - |\mathsf{E}[f|X_1, \ldots, X_{i-1}, X_i = b]| \leq c_i$$

If this condition holds for all possible choices of a and b, and for all i, then Eq. (13.3) follows. What is behind this somewhat contrived definition is the fact that the sequence $Y_i := \mathsf{E}[f|X_1, \ldots, X_{i-1}, X_i]$ is a martingale (the so-called Doob martingale). A martingale is simply a sequence of random variables $Z_0, Z_1, \ldots, Z_n$ such that $\mathsf{E}[Z_i|Z_0, \ldots, Z_{i-1}] = Z_i$, for $i = 1, 2, \ldots, n$. A typical example of a martingale is a uniform random walk in the integer lattice, where a particle can move left, right, up, or down with equal probability. If Z_i denotes the distance of the particle from the origin, the expected distance after one step stays put. A close relative of the Chernoff–Hoeffding bound, known as Azuma's inequality, states that if a martingale sequence $Z_0, Z_1, \ldots, Z_n$ satisfies the bounded difference condition $|Z_i - Z_{i-1}| \leq c_i$ for $i = 1, 2, \ldots, n$, then it is unlikely that Z_n is far from Z_0:

$$\Pr[|Z_n - Z_0| > t] \leq 2e^{-2t^2/\Sigma_i c_i^2} \tag{13.4}$$

In words, if a martingale sequence does not make big jumps, then it is unlikely to stray afar from its starting point. This is true for the random walk; it is very unlikely that after n steps the particle will be far from the origin. Note that for a Doob martingale $Y_0 = \mathsf{E}[f]$ and $Y_n = f$, so that Eq. (13.4) becomes Eq. (13.3).

To see the usefulness of this more awkward formulation, let us drop the assumption that the network is triangle-free and analyze again what happens to the vertex degrees, following the analysis from Ref. [29]. As observed this introduces the problem that the effect of bundles can be very large: changing the value of B_v can affect the new degree by as much as $d-1$. We will therefore accept the fact that the new degree of a vertex is a function of $\Theta(d^2)$ variables, but we will be able to bound the effect of edges at distance one from the vertex. Fix a vertex v and let $N^1(v)$ denote the set of "direct" edges—i.e., the edges incident on v—and let $N^2(v)$ denote the set of "indirect edges" that is, the edges incident on a neighbor of v. Let $N^{1,2}(v) := N^1(v) \bigcup N^2(v)$. Finally, let $T := (T_{e_1}, \ldots, T_{e_m})$, $m = |E(G)|$, be the random vector specifying the tentative color choices of the edges in the graph G. With this notation, the number of edges successfully colored at vertex v is a function $f(T_e, e \in N^{1,2}(v))$ (to study f or the new degree is the same: if f is concentrated so is the new degree).

Let us number the variables so that the direct edges are numbered *after* the indirect edges (this will be important for the calculations to follow). We need to compute

$$\lambda_k := |\mathsf{E}[\mathsf{f} \mid \boldsymbol{T}_{\mathsf{k}-1}, \mathsf{T}_\mathsf{k} = \mathsf{c}_\mathsf{k}] - \mathsf{E}[\mathsf{f} \mid \boldsymbol{T}_{\mathsf{k}-1}, \mathsf{T}_\mathsf{k} = \mathsf{c}'_\mathsf{k}]| \tag{13.5}$$

We decompose f as a sum to ease the computations later. Introduce the indicator functions f_e, $e \in E$: $f_e(\boldsymbol{c})$ is 1 if edge e is successfully colored in coloring $\boldsymbol{c}$, and 0 otherwise. Then $f = \sum_{v \in e} f_e$. Hence we are reduced, by linearity of expectation, to computing for each $e \in N^1(v)$, $|\mathsf{Pr}[f_e = 1 \mid \boldsymbol{T}_{k-1}, T_k = c_k] - \mathsf{Pr}[f_e = 1 \mid \boldsymbol{T}_{k-1}, T_k = c'_k]|$.

To compute a good bound for λ_k in Eq. (13.5), we shall lock together two distributions Y and Y'. Y is distributed as $\boldsymbol{T}$ conditioned on $\boldsymbol{T}_{k-1}$, $T_k = c_k$, and Y', while Y' is distributed as $\boldsymbol{T}$ conditioned on $\boldsymbol{T}_{k-1}$, $T_k = c'_k$. We can think of $\boldsymbol{Y'}$ as identically equal to $\boldsymbol{Y}$ except that $\boldsymbol{Y'}_k = c'_k$. Such a pairing (Y, Y') is called a *coupling* of the two different distributions $[T|\boldsymbol{T}_{k-1}, T_k = c_k]$ and $[T|\boldsymbol{T}_{k-1}, T_k = c'_k]$. It is easily seen that by the independence of all tentative colors, the marginal distributions of $\boldsymbol{Y}$ and $\boldsymbol{Y'}$ are exactly the two conditioned distributions $[\boldsymbol{T} \mid \boldsymbol{T}_{k-1}, T_k = c_k]$ and $[\boldsymbol{T} \mid \boldsymbol{T}_{k-1}, T_k = c'_k]$, respectively. Now let us compute $|\mathsf{E}[\mathsf{f}(\boldsymbol{Y}) - \mathsf{f}(\boldsymbol{Y'})]|$.

First, let us consider the case when $e_1, \ldots, e_k \in N^2(v)$, i.e., only the choices of indirect edges are exposed. Let $e_k = (w, z)$, where w is a neighbor of v. Then, for a direct edge $e \neq vw$, $f_e(\boldsymbol{y}) = f_e(\boldsymbol{y'})$ because in the joint distribution space, $\boldsymbol{y}$ and $\boldsymbol{y'}$ agree on all edges incident on e. So we only need to compute $|\mathsf{E}[\mathsf{f}_{\mathsf{vw}}(\boldsymbol{Y}) - \mathsf{f}_{\mathsf{vw}}(\boldsymbol{Y'})]|$. To bound this simply, we observe first that $f_{vw}(\boldsymbol{y}) - f_{vw}(\boldsymbol{y'}) \in [-1, 1]$ and second that $f_{vw}(\boldsymbol{y}) = f_{vw}(\boldsymbol{y'})$ unless $y_{vw} = c_k$ or $y_{vw} = c'_k$. Thus we can conclude that $\mathsf{E}[\mathsf{f}_{\mathsf{vw}}(\boldsymbol{Y}) - \mathsf{f}_{\mathsf{vw}}(\boldsymbol{Y'})]| \leq \mathsf{Pr}[\mathsf{Y}_\mathsf{e} = \mathsf{c}_\mathsf{k} \vee \mathsf{Y}_\mathsf{e} = \mathsf{c}'_\mathsf{k}] \leq \frac{2}{\mathsf{d}}$.

In fact, one can do a tighter analysis using the same observations. Let us denote $f_e(\boldsymbol{y}, y_{w,z} = c_1, y_e = c_2)$ by $f_e(c_1, c_2)$. Note that $f_{vw}(c_k, c_k) = 0$ and similarly $f_{vw}(c'_k, c'_k) = 0$. Hence

$$\begin{aligned}\mathsf{E}[\mathsf{f}_\mathsf{e}(\boldsymbol{Y}) - \mathsf{f}_\mathsf{e}(\boldsymbol{Y'}) \mid \mathsf{z}] &= (f_{vw}(c_k, c_k) - f_{vw}(c'_k, c_k))\mathsf{Pr}[Y_e = c_k] \\ &\quad + (f_{vw}(c_k, c'_k) - f_{vw}(c'_k, c'_k))\mathsf{Pr}[Y_e = c'_k] \\ &= (f_{vw}(c_k, c'_k) - f_{vw}(c'_k, c_k))\frac{1}{d}\end{aligned}$$

(Here we used the fact that the distribution of colors around v is unaffected by the conditioning around z and that each color is equally likely.) Hence $|\mathsf{E}[\mathsf{f}_\mathsf{e}(\boldsymbol{Y}) - \mathsf{f}_\mathsf{e}(\boldsymbol{Y'})]| \leq \frac{1}{\mathsf{d}}$.

Now let us consider the case when $e_k \in N^1(v)$, i.e., choices of all indirect edges and of some direct edges have been exposed. In this case, we merely observe that f is Lipshitz with constant 2: $|f(\boldsymbol{y}) - f(\boldsymbol{y'})| \leq 2$ whenever $\boldsymbol{y}$ and $\boldsymbol{y'}$ differ in only one coordinate. Hence we can easily conclude that $|\mathsf{E}[\mathsf{f}(\boldsymbol{Y}) - \mathsf{f}(\boldsymbol{Y'})]| \leq 2$.

Overall, $\lambda_k \leq 1/d$ for an edge $e_k \in N^2(v)$, and $\lambda_k \leq 2$ for an edge $e_k \in N^1(v)$. Therefore, we get

$$\sum_k \lambda_k^2 = \sum_{e \in N^2(v)} \frac{1}{d^2} + \sum_{e \in N^1(v)} 4 \leq 4d + 1$$

We thus arrive at the following sharp concentration result by plugging into Eq. (13.3): Let v be an arbitrary vertex and let f be the number of edges successfully colored around v in one stage of the trivial algorithm. Then,

$$\mathsf{Pr}[|f - \mathsf{E}[\mathsf{f}]| > t] \leq 2\exp\left(-\frac{t^2}{2d + \frac{1}{2}}\right)$$

Since $\mathsf{E}[f] = \Theta(d)$, this is a very strong bound.

13.4 Matchings

Maximum matching is probably one of the best studied problems in computer science: given a weighted undirected graph $G = (V, E)$, compute a subset of pairwise nonincident edges (*matching*) of maximum cost. For simplicity, we will focus on the the cardinality version of the problem, where all the edges have weight 1.

It is not hard to show that a maximum matching cannot be computed efficiently (i.e., in polylogarithmic time) in a distributed setting.

Lemma 13.7

Any distributed maximum matching algorithm requires $\Omega(n)$ rounds.

Proof

Consider the following *mailing problem*: let P be a path of $n = 2k + 1$ nodes, and let ℓ and r be the left and right endpoints of the path, respectively. Moreover, let c be the *central* node of the path. Nodes ℓ and r receive the same input bit b, and the problem is to forward b to the central node c. Clearly, this process takes at least k rounds.

Now assume by contradiction that there exists a $o(n)$ distributed maximum matching protocol $\mathcal{M}$. We can use $\mathcal{M}$ to solve the mailing problem above in the following way. All the nodes run $\mathcal{M}$ on the auxiliary graph $P(b)$ obtained from P by removing the edge incident to ℓ if $b = 1$, and the edge incident to r otherwise. If $b = 1$ ($b = 0$), the edge on the left (right) of v must belong to the (unique) maximum matching. This way c can derive the value of the input bit b in $o(n) = o(k)$ rounds, which is a contradiction. □

Fischer et al. [30] described a parallel algorithm to compute a near-optimal matching in arbitrary graphs. Their algorithm can be easily turned into a distributed protocol to compute a $k/(k + 1)$-approximate solution in polylogarithmic time, for any fixed positive integer $k > 0$. A crucial step in the algorithm by Fischer et al. is computing (distributively) a maximal independent set. Since this subproblem is rather interesting by itself in the distributed case, in Section 13.4.1 we will sketch how it can be solved efficiently. In Section 13.4.2 we will describe and analyze the algorithm by Fischer et al.

13.4.1 Distributed Maximal Independent Set

Recall that an *independent set* of a graph is a subset of pairwise nonadjacent nodes. No deterministic protocol is currently known for the problem. Indeed, this is one of the main open problems in distributed algorithms. Luby [31] and independently Alon et al. [32] gave the first distributed randomized algorithms to compute a maximal independent set. Here we will focus on Luby's result, as described in Kozen's book [33].

As the algorithm by Fischer et al., Luby's algorithm was originally thought for a parallel setting, but it can be easily turned into a distributed algorithm. It is worth noticing that transforming an efficient parallel algorithm into an efficient distributed algorithm is not always trivial. For example, there is a deterministic parallel version of Luby's algorithm, while, as mentioned above, no efficient deterministic distributed algorithm is known for the maximal independent set problem.

Luby's algorithm works in stages. In each stage, one (not necessarily maximal) independent set I is computed, and the nodes I are removed from the graph together with all their neighbors. All the edges incident to deleted nodes are also removed. The algorithm ends when no node is left. At the end of the algorithm a maximal independent set is given by the union of the independent sets I computed in the different stages.

It remains to describe how each independent set I is computed. Each node v in the (current) graph independently becomes a *candidate* with probability $\frac{1}{2d(v)}$. Then, for any two adjacent candidates, the one of lower degree is discarded from S (ties can be broken arbitrarily). The remaining candidates form the set I.

Each stage can be trivially implemented with a constant number of communication rounds. The expected number of rounds is $O(\log n)$. More precisely, in each stage at least a constant expected fraction of the (remaining) edges are removed from the graph.

A crucial idea in Luby's analysis is the notion of good nodes: a node v is *good* if at least one-third of its neighbors have degree not larger than v. In particular, this implies

$$\sum_{u \in N(v)} \frac{1}{2d(u)} \geq \frac{1}{6} \tag{13.6}$$

Otherwise v is *bad*. Although there might be few good nodes in a given graph, the edges incident to at least one good node are a lot. Let us call an edge *good* if it is incident to at least one good node, and *bad* otherwise. The following lemma holds.

Lemma 13.8

At least one-half of the edges are good.

Proof

Direct all the edges toward the endpoint of higher degree, breaking ties arbitrarily. Consider any bad edge e directed toward a given (bad) node v. By definition of bad nodes, the out-degree of v is at least twice its own in-degree. Thus we can uniquely map e into a pair of edges (either bad or good) leaving v. Therefore, the edges are at least twice as many as the bad edges. □

Thus it is sufficient to show that in a given stage each good node is removed from the graph with constant positive probability.

Lemma 13.9

Consider a node v in a given stage. Node v belongs to I with probability $\frac{1}{4d(v)}$.

Proof

Let $L(v) = \{u \in N(v) \mid d(u) \geq d(v)\}$ be the neighbors of v of degree not smaller than $d(v)$. Then

$$Pr(v \notin I \mid v \in S) \leq \sum_{u \in L(v)} Pr(u \in S \mid v \in S) = \sum_{u \in L(v)} Pr(u \in S) \leq \sum_{u \in L(v)} \frac{1}{2d(u)} \leq \sum_{u \in L(v)} \frac{1}{2d(v)} \leq \frac{1}{2}$$

Hence $Pr(v \in I) = Pr(v \in I \mid v \in S)\ Pr(v \in S) \geq \frac{1}{2}\frac{1}{2d(v)} = \frac{1}{4d(v)}$. □

Lemma 13.10

Let v be a good node in a given stage. Node v is discarded in the stage considered with probability at least $1/36$.

Proof

We will show that $v \in N(I) = \cup_{u \in I} N(u)$ with probability at least $1/36$. The claim follows. If v has a neighbor u of degree at most 2, by Lemma 13.9, $Pr(v \in N(I)) \geq Pr(u \in I) \geq \frac{1}{4d(u)} = \frac{1}{8}$.

Now assume that all the neighbors of v have degree 3 or larger. It follows that, for every neighbor u of v, $\frac{1}{2d(u)} \leq \frac{1}{6}$. Hence by Eq. (13.6) there exists a subset $M(v)$ of neighbors of v such that $\frac{1}{6} \leq \sum_{u \in M(v)} \frac{1}{2d(u)} \leq \frac{1}{3}$. Thus

$$\begin{aligned}
Pr(v \in N(I)) &\geq Pr(\exists\, u \in M(v) \cap I) \\
&\geq \sum_{u \in M(v)} Pr(u \in I) - \sum_{u,w \in M(v),\ u \neq w} Pr(u \in I \wedge w \in I) \\
&\geq \sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u,w \in M(v),\ u \neq w} Pr(u \in S \wedge w \in S) \\
&\geq \sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u,w \in M(v),\ u \neq w} Pr(u \in S) Pr(w \in S) \\
&\geq \sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u \in M(v)} \sum_{w \in M(v)} \frac{1}{2d(u)} \frac{1}{2d(w)} \\
&= \left(\frac{1}{2} - \sum_{w \in M(v)} \frac{1}{2d(w)}\right) \sum_{u \in M(v)} \frac{1}{2d(u)} \geq \left(\frac{1}{2} - \frac{1}{3}\right)\frac{1}{6} = \frac{1}{36}
\end{aligned}$$

□

13.4.2 The Distributed Maximum Matching Algorithm

Consider an arbitrary matching M of a graph $G = (V, E)$. A node is *matched* if it is the endpoint of some edge in M, and *free* otherwise. An *augmenting path* P with respect to M is a path (of odd length) whose endpoints are free and whose edges are alternatively inside and outside M. The reason of the name is that we can obtain a matching M' of cardinality $|M| + 1$ from M, by removing from M all the edges which are also in P, and by adding to M the remaining edges of P (in other words, M' is the symmetric difference $M \oplus P$ of M and P).

The algorithm by Fischer et al. is based on the following two lemmas by Hopcroft and Karp [34]. Let two paths be *independent* if they are node-disjoint. Note that a matching can be augmented along several augmenting paths simultaneously, provided that such paths are independent.

Lemma 13.11

If a matching is augmented along a maximal set of independent shortest augmenting paths, then the shortest augmenting paths length grows.

Lemma 13.12

Suppose a matching M does not admit augmenting paths of length $2k - 1$ *or smaller. Then the size of M is at least a fraction* $\frac{k}{k+1}$ *of the maximum matching size.*

Proof

Let M^* be a maximum matching. The symmetric difference $M' = M \oplus M^*$ contains $|M^*| - |M|$ independent augmenting paths with respect to M. Since each of these paths contains at least k edges of M, $|M^*| - |M| \leq |M|/k$. The claim follows. □

We are now ready to describe and analyze the approximate maximum matching algorithm by Fischer et al. The algorithm proceeds in stages. In each stage i, $i \in \{1, 2, \ldots, k\}$, the algorithm computes a maximal independent set P_i of augmenting paths of length $2i - 1$ with respect to the current matching M. Then M is augmented according to P_i. Stage i can be implemented by simulating Luby's algorithm on the auxiliary graph induced by the augmenting paths considered, where the nodes are the paths and the edges are the pairs of nonindependent paths. In particular, Luby's algorithm takes $O(\log n^{2i})$ rounds in expectation in the auxiliary graph, where each such round can be simulated within $O(i)$ rounds in the original graph. Note that, by Lemma 13.11, at the end of stage i there are no augmenting paths of length $2i - 1$ or smaller. It follows from Lemma 13.12 that at the end of the kth stage the matching computed is $\frac{k}{k+1}$-approximate. The total expected number of rounds is trivially $O(k^3 \log n)$. The following theorem summarizes the discussion above.

Theorem 13.1

For every integer $k > 0$, *there is a distributed algorithm which computes a matching of cardinality at least* $\frac{k}{k+1}$ *times the maximum matching cardinality within* $O(k^3 \log n)$ *communication rounds in expectation.*

Wattenhofer and Wattenhofer [35] gave a $O(\log^2 n)$ randomized algorithm to compute a constant approximation in the weighted case. In the deterministic case weaker results are available. This is mainly due to the fact that we are not able to compute maximal independent sets deterministically. Hańćkowiak et al. [36,37] described an efficient distributed deterministic algorithm to compute a maximal matching. Recall that any maximal matching is a 2-approximation for the maximum matching problem. Recently, a 1.5 deterministic distributed approximation algorithm was described in Ref. [38].

13.5 LP-Based Distributed Algorithms

It might come as a surprise that LP-based methods find their application in a distributed setting. In this section we describe some primal-dual algorithms for vertex cover problems that give "state-of-the-art" approximations. In general, it seems that the primal-dual method, one of the most successful techniques in approximation algorithms, when applied to graph algorithms exhibits "local" properties that makes it amenable to a distributed implementation. The best way to explain what we mean is to work out an example.

We will illustrate the method by considering the *vertex cover* problem: given an undirected graph $G = (V, E)$, with positive weights $\{c(v)\}_{v \in V}$, compute a minimum-cost subset V' of nodes such that each edge is incident to at least one node in V'. This *NP*-hard problem is approximable within 2 [39], and not approximable within 1.1666 unless P = NP [40]. In the centralized case there is a primal-dual 2-approximation algorithm. The distributed implementation we give yields a $2 + \epsilon$ approximation, where ϵ can be any fixed constant greater than 0. The number of communication rounds of the algorithm is $O(\log n \log \frac{1}{\epsilon})$.

The sequential primal-dual algorithm works as follows. We formulate the problem as an integer program (IP):

$$\begin{aligned} \min \quad & \sum_{v \in V} c(v) \cdot x_v && && \text{(IP)} \\ \text{s.t.} \quad & x_v + x_u \geq 1 && \forall e = (u, v) \in E && (13.7) \\ & x_v \in \{0, 1\} && \forall v \in V && (13.8) \end{aligned}$$

The binary indicator variable x_v, for each $v \in V$, takes value 1 if $v \in V'$, and 0 otherwise.

We now let (LP) be the standard LP relaxation obtained from (IP) by replacing the constraints (13.8) by $x_v \geq 0$ for all $v \in V$. In the linear programming dual of (LP) we associate a variable α_e with constraints (13.7) for every $e \in E$. The linear programming dual (D) of (LP) is then

$$\begin{aligned} \max \quad & \sum_{e \in E} \alpha_e && && \text{(D)} \\ \text{s.t.} \quad & \sum_{e=(u,v) \in E} \alpha_e \leq c(v) && \forall v \in V && (13.9) \\ & \alpha_e \geq 0 && \forall e \in E && (13.10) \end{aligned}$$

The starting primal and dual solutions are obtained by setting to 0 all the variables x_v and α_e. Observe that the dual solution is feasible while the primal one is not. We describe the algorithm as a continuous process. We let all the variables α_e grow at uniform speed. As soon as one constraint of type (13.9) is satisfied with equality (it becomes *tight*), we set the corresponding variable x_v to 1, and we freeze the values α_e of the edges incident to v. The α-values of frozen edges do not grow more, so that the constraint considered remains tight. The process continues until all edges are frozen. When this happens the primal solution becomes feasible. To see why, suppose not. But then there is an edge $e = uv$ which is not covered, i.e., $x_u = x_v = 0$. This means that the constraints corresponding to u and v are not tight and α_e can continue to grow, a contradiction.

Thus the set $V' := \{u : x_u = 1\}$ is a cover. Its cost is upper-bounded by twice the cost of the dual solution:

$$\sum_{v \in V} c(v)\, x_v = \sum_{v \in V'} c(v) \leq \sum_{v \in V'} \sum_{e=(u,v) \in E} \alpha_e \leq 2 \sum_{e \in E} \alpha_e$$

Thus the solution computed is 2-approximate by weak duality.

The continuous process above can be easily turned into a discrete one. Let $c'(v)$ be the difference between the right- and the left-hand side of constraints (13.9) in a given instant of time (*residual weight*):

$$c'(v) = c(v) - \sum_{e=(u,v) \in E} \alpha_e$$

Moreover, let $d'(v)$ be the current number of nonfrozen (*active*) edges incident to v. The idea is to raise in each step the dual value α_e of all the active edges by the minimum over all nodes v such that $x_v = 0$ of the quantity $c'(v)/d'(v)$. This way, in each step at least one extra node enters the vertex cover.

There is a simple-minded way to turn the algorithm above into a distributed algorithm: each node v maintains the quantities $c'(v)$ and $d'(v)$. A node is *active* if $c'(v) > 0$ and $d'(v) > 0$, that is if v and at least one of its neighbors are not part of the vertex cover. In each round each active node v sends a *proposal* $c'(v)/d'(v)$ to all its active neighbors. Then it decreases $c'(v)$ by the minimum of all the proposals sent and received. If $c'(v)$ becomes 0, v enters the vertex cover. Otherwise, if $d'(v)$ becomes 0, v halts since all its neighbors already belong to the vertex cover.

The main drawback of this approach is that it is very slow. In fact, it may happen that in each step a unique node enters the vertex cover, thus leading to a linear number of rounds. Khuller et al. [41] showed how to circumvent this problem by losing something in the approximation. Here we will present a simplified version of their algorithm and analysis (which was originally thought for weighted set cover in a parallel setting). The idea is to slightly relax the condition for a node v to enter the vertex cover: it is sufficient that the residual weight $c'(v)$ falls below $\epsilon\, c(v)$, for a given (small) constant $\epsilon > 0$.

Theorem 13.2

The algorithm above computes a $\frac{2}{1-\epsilon}$-approximate vertex cover within $O(\log n \log \frac{1}{\epsilon})$ rounds.

Proof

The bound on the approximation easily follows by adapting the analysis of the primal-dual centralized approximation algorithm:

$$(1-\epsilon)\, apx = \sum_{v \in V'} (1-\epsilon)\, c(v) \leq \sum_{v \in V'} \sum_{e=(u,v) \in E} \alpha_e \leq 2 \sum_{e \in E} \alpha_e \leq 2\, opt$$

To bound the number of rounds we use a variant of the notion of good nodes introduced in Section 13.4.1. Consider the graph induced by the active nodes in a given round, and call the corresponding edges *active*. Let us direct all the active edges toward the endpoint which makes the smallest proposal. A node is *good* if its in-degree is at least one-third of its (total) degree. By basically the same argument as in Section 13.4.1, at least one-half of the edges are incident to good nodes. Moreover, the residual weight of a node which is good in a given round decreases by at least one-third in the round considered. As a consequence, a node can be good in at most $\log_{3/2} \frac{1}{\epsilon}$ rounds (after those many rounds it must enter the vertex cover).

We will show next that the total number of active edges halves every $O(\log \frac{1}{\epsilon})$ rounds by means of a potential function argument. It follows that the total number of rounds is $O(\log m \log \frac{1}{\epsilon}) = O(\log n \log \frac{1}{\epsilon})$. Let us associate $2 \log_{3/2} \frac{1}{\epsilon}$ credits to each edge, and thus $2m \log_{3/2} \frac{1}{\epsilon}$ credits to the whole graph. When a node v is good in a given step, we remove one credit from each edge incident to it. Observe that an active edge e in a given round must have at least two credits left. This is because otherwise one of the endpoints of e would already belong to the vertex cover, and thus e could not be active. By m_j we denote the number of active edges in round j. Recall that in each round at least one-half of the edges are incident to a good node, and such edges loose at least one credit each in the round considered. Thus the total number of credits in round j decreases by a quantity g_j which satisfies $g_j \geq m_j/2$. Consider an arbitrary round i, and let k be the smallest integer such that $m_{i+k} < m_i/2$ (or $i+k$ is the last round). It is sufficient to show that $k = O(\log \frac{1}{\epsilon})$. In each round j, $j \in \{i, i+1, \ldots, i+k-1\}$, the number of edges satisfies $m_j \geq m_i/2$. The total number of credits at the beginning of round i is at most $2m_i \log_{3/2} \frac{1}{\epsilon}$, and the algorithm halts when no credit is left. Therefore,

$$2m_i \log_{3/2} \frac{1}{\epsilon} \geq \sum_{j=i}^{i+k-1} g_j \geq \sum_{j=i}^{i+k-1} \frac{m_j}{2} \geq \sum_{j=i}^{i+k-1} \frac{m_i}{4} = k\frac{m_i}{4} \quad \Rightarrow \quad k \leq 8 \log_{3/2} \frac{1}{\epsilon} = O\left(\log \frac{1}{\epsilon}\right) \quad \square$$

By choosing $\epsilon = 1/(nC+1)$, where C is the maximum weight, the algorithm by Khuller et al. computes a 2-approximate vertex cover within $O(\log n \log(nC))$ rounds. Recently, Grandoni et al. [42] showed how to achieve the same task in $O(\log(nC))$ rounds by means of randomization. They reduce the problem to the computation of a maximal matching in an auxiliary graph of nC nodes (to have an idea of the

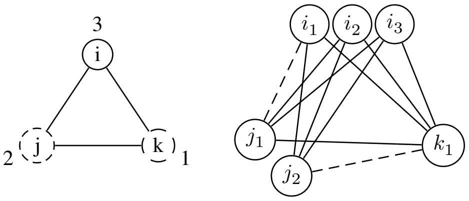


FIGURE 13.2 A weighted graph G (on the left) with the corresponding auxiliary graph $\widetilde{G}$. A maximal matching M of $\widetilde{G}$ is indicated via broken lines. The nodes of G such that all the corresponding nodes in $\widetilde{G}$ are matched form a 2-approximate vertex cover.

reduction, see Figure 13.2). Such matching can be computed in $O(\log(nC))$ rounds via the randomized, distributed maximal matching algorithm by Israeli and Itai [43]. The authors also show how to keep small the message size and the local computation time by computing the matching *implicitly*.

The *capacitated* vertex cover problem is the generalization of the vertex cover problem where each node v can cover only a limited number $b(v) \leq d(v)$ of edges incident to it. Grandoni et al. [44] showed how to compute within $O(\frac{\log nC}{\epsilon})$ rounds an $(2 + \epsilon)$-approximate solution, if any, which violates the capacity constraints by a factor at most $(4 + \epsilon)$. They also proved that any distributed constant approximation algorithm must violate the capacity constraints by a factor at least 2. This, together with the known lower bounds on the approximation of (classical) vertex cover, shows that their algorithm is the best possible modulo constants. The algorithm by Grandoni et al. builds up on a primal-dual centralized algorithm developed for the purpose, which computes a 2 approximation with a factor 2 violation of the capacity constraints. Turning such primal-dual algorithm into a distributed protocol is far more involved than in the case of classical vertex cover.

13.6 What Can and Cannot Be Computed Locally?

This fundamental question in distributed computing was posed by Naor and Stockmeyer [45]. Here, "locally" means that the nodes of the network use information available locally from a neighborhood that can be reached in time much smaller than the size of the network. For many natural distributed network problems such as leader election and consensus the parameter determining the time complexity is not the number of vertices, but the network *diameter* D, which is the maximum distance (number of hops) between any two nodes [46]. A natural question is whether other fundamental primitives can be computed in $O(D)$ time in a distributed setting. If the model allows messages of unbounded size, then there is a trivial affirmative answer to this question: collect all the information at one vertex, solve the problem locally and then transmit the result to all vertices. The problem is therefore only interesting in the more realistic model where we assume that each link can transmit only B bits in any time step (B is usually taken to be a constant or $O(\log n)$).

A landmark negative result in this direction was that of Linial [47] which investigated the time complexity of various global functions of a graph computed in a distributed setting. Suppose that n processors are arranged in a ring and can communicate only with their immediate neighbors. Linial showed that a three-coloring of the n-cycle requires time $\Omega(\log^* n)$. This result was extended to randomized algorithms by Naor [48]: any probabilistic algorithm for three-coloring the ring must take at least $\frac{1}{2}\log^* n - 2$ rounds, otherwise the probability that all processors are colored legally is less than $\frac{1}{2}$. The bound is tight (up to a constant factor) in light of the deterministic algorithms of Cole and Vishkin [49].

There has been surprisingly little continuation of work in this direction until fairly recently. Garay et al. [50] gave an algorithm of complexity $O(D + \sqrt{n}\log n)$ to compute a minimum spanning tree (MST) of a graph on n vertices with diameter D. Similar bounds were attained by other methods, but none managed to break the $\sqrt{n}$ barrier, leading to the suspicion that it might be impossible to compute the MST in time $o(\sqrt{n})$ and so this problem is fundamentally harder than the other paradigm problems. The issue was finally settled by Peleg and Rubinovich [51], who showed a $\Omega(\sqrt{n})$ lower bound on the problem (up to log factors). Subsequently, Elkin [52] improved the lower bound and also extended it to distributed approximation algorithms. Kuhn et al. [53] gave lower bounds on the complexity of computing the minimum vertex cover (MVC) and the MDS of a graph: in k communication rounds, the MVC and MDS can only be approximated to factors of $\Omega(n^{ck^2}/k)$ and $\Omega(\Delta^{1/k}/k)$ (where Δ is the maximum degree of the graph). Thus, the number of rounds required to reach a constant or even a polylog approximation is at least $\Omega(\sqrt{\log n/\log\log n})$ and $\Omega(\log\Delta/\log\log\Delta)$. The same lower bounds also apply to the construction of maximal matchings and maximal independent sets via a simple reduction.

13.6.1 A Case Study: Minimum Spanning Tree

Here, we give a self-contained exposition of the lower bound for the MST problem due to Refs. [51,52]. We will give the full proof of a bound somewhat weaker than the optimal result of Elkin to convey the underlying ideas more clearly. The basic idea is easy to explain using the example of Peleg and Rubinovich [51] (see Figure 13.3). The network consists of m^2 country road and one highway. Each country road has m toll stations and between every two successive toll station are m towns. The highway has m toll stations with no towns in between. Each toll station number i on each country road is connected to the corresponding highway toll station. The left end of the country road i is labelled s_i and its right end r_i. The left end of the highway is labelled s and the right end r. This is the basic underlying graph. Note that there are $\Theta(m^4)$ vertices and the diameter is $\Theta(m)$.

As for the weights, every edge along the highway or on the country roads has weight 0. The roads connecting the toll stations on the country roads to the corresponding toll stations on the highway have weight ∞ *except for the first and last toll stations.* The toll station connections at the right end between each r_i and r are all 1. At the left end, between each s_i and s, they take either the value 0 or ∞.

What does the MST of this network look like? First, we may as well include the edges along the highway and each path since these have zero cost. Also the intermediate connecting edges have weight ∞ and so are excluded. That leaves us with the connecting edges on the left and on the right. The choice here depends on the weights on the left connecting edges. There are m connecting edges from the left vertex s. If the edge (s, s_i) has weight ∞, then we must exclude this and include the matching connection (r_i, r) at the right end. In contrast, if edge (s, s_i) has weight 0, then we must include this and exclude the corresponding edge (r_i, r) at the right to r. Thus there are m^2 decisions made at s depending on the weights of the corresponding edges, and these decisions must be conveyed to r to pick the corresponding complementary edges. How quickly can these m^2 bits be conveyed from s to r? Clearly, it would take very long to route along the country roads, and so one must use the highway edges instead. Each highway edge can forward only B bits at any time step. So, heuristically, transporting the m^2 bits takes $\Omega(m^3/B)$ steps.

To make this heuristic argument formal, Peleg and Rubinovich introduced a *mailing problem* to be solved on a given network. In the example above, the *sender* s has m^2 bits that need to be transported to the *receiver* r. At each step one can forward B bits along any edge. How many steps do we need to correctly route the m^2 bits from the sender to the receiver? It is easy to see that there is a reduction from the mailing problem to that of computing the MST: for each of the input bits at s, set the weights on the connecting edges accordingly: the weight (s, s_i) is ∞ if the input bit i is 1 and 0 otherwise. Now compute the MST. Then, if vertex r notices that the edge (r_i, r) is picked in the MST, it decodes i

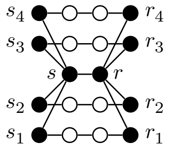


FIGURE 13.3 MST lower bound graph for $m = 2$. The black nodes are the toll stations and the white nodes are the towns.

as 1 and as 0 otherwise. This will correctly solve the mailing problem, due to the structure of the MST discussed above. Thus, a lower bound on the mailing problem implies the same lower bound on the MST problem.

In fact by a slight change, the correspondence can be extended from exact to approximation algorithms. Elkin [52] introduced the *corrupted mail* problem. Here there are Γ bits at the sender exactly $\alpha\Gamma$ of which are 1's, where α and Γ are parameters. In the example above, $\Gamma = m^2$. The receiver should get Γ bits delivered to it, but these are allowed to be somewhat corrupted. The restrictions are (a) any input bit that was 1 must be transmitted correctly without corruption and (b) the total number of 1's delivered can be at most $\beta\Gamma$, where $\beta \geq \alpha$ is another parameter. Consider solving the (α, β) corrupted mail problem on the Peleg–Rubinovich example. As in the reduction before, the vertex s sets the weights on the left connections according to its input and so exactly $\alpha\Gamma$ connections have weight ∞, and the rest 0. The optimal MST has weight exactly $\alpha\Gamma$ obtained by picking the corresponding right connections. Now, instead of the optimal MST, suppose we apply a protocol to compute a β/α approximation. This approximate MST can have weight at most $\beta\Gamma$, and it must include the connection edges at r paired with the infinite weight edges at s. Thus, if r sets its bits as before corresponding to which of its connections are in the approximate MST, we get a correct protocol for the (α, β) corrupted mail problem. Thus a lower bound for the (α, β) corrupted mail problem implies the same lower bound for a $\frac{\beta}{\alpha}$-approximate MST.

We are thus left with the task of proving a lower bound for the corrupted mail problem. Let the *state* $\psi(v, t)$ of a vertex v at some time t denote the sequence of messages it has received up to this time. Consider the start vertex s at time 0: this can be in any of $\binom{\Gamma}{\alpha\Gamma}$ states corresponding to the input it receives. At this time, on the other hand, the vertex r (and indeed, any other vertex) is in a fixed state (having received no messages at all). As time progresses and messages are passed, the set of possible states that other vertices are in expands. Eventually, the set of possible states that vertex r is in must be large enough to accommodate the output corresponding to all the possible inputs at s. Each possible state of r with at most $\beta\Gamma$ 1's can be the correct answer to at most $\binom{\beta\Gamma}{\alpha\Gamma}$ input configurations at s. Hence, the set of output states at r must be at least $\binom{\Gamma}{\alpha\Gamma}/\binom{\beta\Gamma}{\alpha\Gamma} \geq (1/e\beta)^{\alpha\Gamma}$.

Now, we will argue that it must take a long time for any protocol, before enough messages arrive at r for the set of its possible states to have this size. Consider the *tail sets* T_i, $i \geq 1$ which consist of the tail of each country road from vertex i until the end, and the corresponding fragment of the highway consisting of the vertices $h_{\lceil i/m \rceil m}$ until h_{m^2}. Also, set $T_0 := V \setminus \{h_0\}$. For a subset of vertices U, let $\mathcal{C}(U, t)$ denote the set of all possible vectors of states of the vertices in U at time t, and let $\rho(U, t) := |\mathcal{C}(U, t)|$. Note that $\rho(T_0, 0) = 1$ although $\rho(\{s\}, 0) = \binom{\Gamma}{\alpha\Gamma}$.

We now focus on how set of configurations of the tail sets T_i grow in time. Fix a configuration $C \in \mathcal{C}(T_t, t)$. How many configurations in $\mathcal{C}(T_{t+1}, t+1)$ can this branch into? The tail set T_{t+1} is connected to the rest of the graph by one highway edge f and by m^2 path edges. Each of the path edges carries a unique message determined by the state of the left endpoint in configuration C. The state of the left endpoint of the highway edge f is not determined by C and hence there could be a number of possible messages that could be relayed along it. However, because of the restriction that at most B bits can be transmitted along an edge at any time step, the total number of possible behaviors observable on edge f at this time step is at most $2^B + 1$. Thus the configuration C can branch off into at most $2^B + 1$ possible configurations $C' \in \mathcal{C}(T_{t+1}, t+1)$. Thus we have argued that for $0 \leq t < m^2$, $\rho(T_{t+1}, t+1) \leq (2^B + 1)\rho(T_t, t)$. By induction, this implies that for $0 \leq t < m^2$, $\rho(T_t, t) \leq (2^B + 1)^t$. Thus finally, we have, that if t^* is the time at which the protocol ends, then either $t^* \geq m^2$, or $(1/e\beta)^{\alpha\Gamma} \leq \rho(\{r\}, t^*) \leq \rho(T_{t^*}, t^*) \leq (2^B + 1)^{t^*}$. Hence, $t^* \geq \min(m^2, \alpha\Gamma \log(\frac{1}{e\beta})/(B + 1))$.

Recalling that $\Gamma = m^2$ in our specific graph, and taking β to be a constant such that $\beta e < 1$, $t^* = \Omega(\alpha m^2/B)$, or, in terms of the number of vertices $n = \Theta(m^4)$ of the graph, $t^* = \Omega(\alpha\sqrt{n}/B)$. If we have a $H := \beta/\alpha$ approximation algorithm for the MST, this implies that $t^* = \Omega(\sqrt{n}/HB)$, implying the trade-off $t^* H = \Omega(\sqrt{n}/B)$ between time and approximation. Elkin [52] improves the lower bound for t^* to $t^* = \Omega\left(\sqrt{n/B}/H\right)$, implying the time-approximation trade-off $t^{*2}H = \Omega\left(\sqrt{n/B}\right)$, and gives a protocol achieving this trade-off.

13.6.2 The Role of Randomization in Distributed Computing

Does randomization help in a distributed setting? This is a fundamental open question in distributed computing. For some of the problems discussed, such as three-coloring on a ring, we have noted that matching lower bounds hold for randomized algorithms. By the usual application of Yao's Minimax Theorem, Elkin's lower bound also applies to randomized algorithms. For the problem of computing maximal matchings and maximal independent sets, there are simple randomized algorithms, whereas the result of Kuhn et al. [53] shows a superpolylog lower bound for deterministic algorithms. A classification of problems by the degree to which randomization helps is an interesting open problem.

References

[1] Basagni, S., Mastrogiovanni, M., Panconesi, A., and Petrioli, C., Localized protocols for ad hoc clustering and backbone formation: A performance comparison, *IEEE Trans. on Parallel and Dist. Systems,* 17(4), 292, 2006.

[2] Rajagopalan, S. and Vazirani, V. V., Primal-dual RNC approximation algorithms for set cover and covering integer programs, *SIAM J. Comput.*, 28(2), 525(electronic), 1999.

[3] Vazirani, V. V., *Approximation Algorithms*, Springer, Berlin, 2001.

[4] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M., *Complexity and Approximation*, Springer, Berlin, 1999.

[5] Raz, R. and Safra, S., A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP, *Proc. of STOC*, 1997, p. 475.

[6] Arora, S. and Sudan, M., Improved low-degree testing and its applications, *Combinatorica*, 23(3), 365, 2003.

[7] Feige, U., A threshold of ln n for approximating set cover, *JACM*, 45(4), 634, 1998.

[8] Dubhashi, D., Mei, A., Panconesi, A., Radhakrishnan, J., and Srinivasan, A., Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons, *Proc. of SODA,* 2003, p. 717.

[9] Jia, L., Rajaraman, R., and Suel, T., An efficient distributed algorithm for constructing small dominating sets, *Dist. Comput.*, 15, 193, 2002.

[10] Kuhn, F., and Wattenhofer, R., Constant-time distributed dominating set approximation, *Dist. Comput.*, 17(4), 303–310, 2005.

[11] Karp, R. M., Probabilistic recurrence relations, *JACM*, 41(6), 1136, 1994.

[12] Dubhashi, D. and Panconesi, A., Concentration of measure for the analysis of randomization algorithms. Cambridge University Press. Forthcoming. http://www.dsi.uniroma1.it/~ale/Papers/master.pdf

[13] Matoušek. J., *Lectures on Discrete Geometry*, Graduate Texts in Mathematics, Vol. 212, Springer, Berlin, 2002.

[14] Bellare, M., Goldreich, O., and Sudan, M., Free bits, pcps and non-approximability—towards tight results, *SIAM J. Comput.*, 27, 804, 1998.

[15] Feige, U. and Kilian, J., Zero knowledge and the chromatic number, *JCSS*, 57, 187, 1998.

[16] Halldòrsson, M. M., A still better performance guarantee for approximate graph coloring, *Inform. Proc. Lett.*, 45, 19, 1993.

[17] Johansson, O., Simple distributed $\delta + 1$-coloring of graphs, *Inform. Proc. Lett.*, 70(5), 229, 1999.

[18] Luby, M., Removing randomness in parallel computation without a processor penalty, *JCSS*, 47(2), 250, 1993.

[19] Finocchi, I., Panconesi, A., and Silvestri, R., An experimental study of simple, distributed vertex colouring algorithms, *Proc. of SODA*, 2002.

[20] Czygrinow, A., Hańćkowiak, M., and Karoński, M., Distributed O(Delta log(n))-edge-coloring algorithm, *Proc. Eur. Symp. on Algorithms*, 2001, p. 345.

[21] Bollobas, B., *Graph Theory: An Introductory Course*, Springer, Berlin, 1979.

[22] Dubhashi, D., Grable, D., and Panconesi, A., Nearly-optimal, distributed edge-colouring via the nibble method, *Theor. Comp. Sci.*, 203, 225, 1998.

[23] Grable, D. and Panconesi, A., Nearly optimal distributed edge colouring in $O(\log\log n)$ rounds, *Random Struct. Algorithms*, 10(3), 385, 1997.

[24] Bollobas, B., Chromatic number, girth and maximal degree, *SIAM J. Disc. Math.*, 24, 311, 1978.

[25] Grable, D. and Panconesi, A., Fast distributed algorithms for brooks-vizing colourings, *Proc. of SODA*, 1998, p. 473.

[26] McDiarmid, C., Concentration. In Habib, M., McDiarmid, C., Ramirez-Alfonsin, J., and Reed, B., Eds., *Probabilistic Methods for Algorithmic Discrete Mathematics*, Springer, New York, 1998, 195–248.

[27] Molly, M. S. O. and Reed, B. A., A bound on the strong chromatic index of a graph. *J. Comb. Theory*, Ser. B 69(2), 103–109, 1997.

[28] Mitzenmacher, M. and Upfal, E., *Probability and Computing*, Cambridge University Press, Cambridge, 2005.

[29] Dubhashi, D. P., Martingales and locality in distributed computing, FSTTCS 1998, 174–185.

[30] Fischer, T., Goldberg, A. V., Haglin, D. J., and Plotkin, S., Approximating matchings in parallel, *Inf. Proc. Lett.*, 46, 115, 1993.

[31] Luby, M., A simple parallel algorithm for the maximal independent set problem, *Proc. of STOC*, 1985, p. 1.

[32] Alon, N., Babai, L., and Itai, A., A fast and simple randomized parallel algorithm for the maximal independent set problem, *J. Algorithms*, 7, 567, 1986.

[33] Kozen, D., *The Design and Analysis of Algorithms*, Springer, Berlin, 1992.

[34] Hopcroft, J. E. and Karp, R. M., An $n^{5/2}$ algorithm for maximum matching in bipartite graphs, *SIAM J. Comput.*, 2, 225, 1973.

[35] Wattenhofer, M. and Wattenhofer, R., Distributed weighted matching, *Proc. Intl. Symp. on Dist. Comput.*, 2004, p. 335.

[36] Hańćkowiak, M., Karoński, M., and Panconesi, A., On the distributed complexity of computing maximal matchings, *Proc. of SODA*, 1998, p. 219.

[37] Hańćkowiak, M., Karoński, M., and Panconesi, A., On the distributed complexity of computing maximal matchings, *SIAM J. Discrete Math.*, 15(1), 41, 2001.

[38] Czygrinow, A., Hańćkowiak, M., and Szymanska, E., A fast distributed algorithm for approximating the maximum matching, *Proc. Eur. Symp. on Algorithms*, 2004, p. 252.

[39] Monien, B. and Speckenmeyer, E., Ramsey numbers and an approximation algorithm for the vertex cover problem, *Acta Informatica*, 22, 115, 1985.

[40] Håstad, J., Some optimal inapproximability results, *Proc. of STOC*, 1997, p. 1.

[41] Khuller, S., Vishkin, U., and Young, N., A primal-dual parallel approximation technique applied to weighted set and vertex cover, *J. Algorithms*, 17(2), 280, 1994.

[42] Grandoni, F., Könemann, J., and Panconesi, A., Distributed weighted vertex cover via maximal matchings, *Intl. Comput. and Comb. Conf.*, 2005.

[43] Israeli, A. and Itai, A., A fast and simple randomized parallel algorithm for maximal matching, *Inf. Proc. Lett.*, 22, 77, 1986.

[44] Grandoni, F., Könemann, J., Panconesi, A., and Sozio, M., Primal-dual based distributed algorithms for vertex cover with semi-hard capacities, *Symp. on Principles of Dist. Comput.*, 2005, p. 118.

[45] Naor, M. and Stockmeyer, L., What can be computed locally? *SIAM J. Comput.*, 24(6), 1259, 1995.

[46] Peleg, D., *Distributed Computing*, SIAM Monographs on Disc. Math. and Appl., Vol. 5, 2000.

[47] Linial, N., Locality in distributed graph algorithms, *SIAM J. Comput.*, 21(1), 193, 1992.

[48] Naor, M., A lower bound on probabilistic algorithms for distributive ring coloring, *SIAM J. Disc. Math.*, 4(3), 409, 1991.

[49] Cole, R. and Vishkin, U., Deterministic coin tossing with applications to optimal parallel list ranking, *Inf. Control*, 70(1), 32, 1986.

[50] Garay, J. A., Kutten, S., and Peleg, D., A sublinear time distributed algorithm for minimum-weight spanning trees, *SIAM J. Comput.*, 27(1), 302, 1998.

[51] Peleg, D. and Rubinovich, V., A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction, *SIAM J. Comput.*, 30(5), 1427, 2000.

[52] Elkin, M., Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem, *Proc. of STOC*, 2004, p. 331.

[53] Kuhn, F., Moscibroda, T., and Wattenhofer, R., What cannot be computed locally! *Proc. Symp. on Principles of Dist. Comput.*, 2004.

14

Empirical Analysis of Randomized Algorithms

Holger H. Hoos
University of British Columbia

Thomas Stützle
Free University of Brussels

14.1 Introduction

Heuristic algorithms are often difficult to analyse theoretically; this holds in particular for advanced, randomised algorithms that perform well in practice, such as high-performance stochastic local search (SLS) procedures (also known as metaheuristics) [1]. Furthermore, for various reasons, the practical applicability of the theoretical results that can be achieved is often very limited. Some theoretical results are obtained under idealised assumptions that do not hold in practical situations—as is the case, for example, for the well-known convergence result for simulated annealing [2]. Also, most complexity results apply to worst-case behaviour, and average-case results, which are fewer and typically much harder to prove, are often based on instance distributions that are unlikely to be encountered in practice. Finally, theoretical bounds on the run times of heuristic algorithms are typically asymptotic and do not reflect the actual behaviour accurately enough for many purposes, in particular, for comparative performance analyses. For these reasons, researchers (and practitioners) typically use empirical methods when analysing or evaluating heuristic algorithms.

In many ways, the issues and considerations arising in the empirical analysis of algorithmic behaviour are quite similar to those commonly encountered in experimental studies in biology, physics or any other empirical science. Fundamentally, to investigate a complex phenomenon of interest, the classical scientific cycle of observation, hypothesis, prediction and experiment is followed to obtain a model that explains the phenomenon. Different from natural phenomena, algorithms are completely specified and mathematically defined at the lowest level; still, in many cases, this knowledge is insufficient for theoretically deriving all relevant aspects of their behaviour. In this situation, empirical approaches, based on computational experiments, are often not only the sole way of assessing a given algorithm, but also have the potential to provide insights into practically relevant aspects of algorithmic behaviour that appear to lie well beyond the reach of theoretical analysis.

Some general goals are common to all empirical studies: *Reproducibility* ensures that experiments can be repeated with the same outcome; it requires that all relevant experimental conditions and protocols are specified clearly and in sufficient detail. In the empirical analysis of algorithmic behaviour, reproducibility is greatly facilitated by the fact that actual computations can in principle be replicated exactly. However, complications can arise when dealing with randomised algorithms or randomly generated input data, in which case statistical significance and sample sizes can become critical issues (despite the fact that typically, pseudo-random number generators are used to implement random processes). *Comparability* with past and future related results ensure that empirical results are useful in the context of larger scientific endeavours. To achieve this goal, experiments have to be designed in such a way that their results can be meaningfully compared to those from relevant previous works and facilitate comparisons with related results expected from future experiments. Finally, perhaps the main goal of any empirical study is to gain *insight and understanding*; this implies that experiments should be designed in such a way that their outcome is likely to shed light on important, previously open questions regarding the phenomenon of interest. In the empirical analysis of algorithms, in many cases these questions are of the form 'Algorithm A has property X', and in particular, 'Algorithm A performs better than Algorithm B'.

14.2 Decision Algorithms

Many computational problems take the form of *decision problems*, in which solutions are characterised by a set of logical conditions. As an example, consider the following decision variant of the travelling salesman problem (TSP): given an edge-weighted graph and a real number b, does there exist a Hamiltonian cycle (i.e., a round trip that visits every vertex exactly once) with total weight at most b? Other well-known examples of decision problems include the propositional satisfiability problem (SAT), the graph colouring problem and certain types of scheduling problems.

A *decision algorithm* is an algorithm that takes as an input an instance of a given decision problem and determines whether the instance is *soluble*, i.e., whether it has a solution. In most cases, if a solution is found, that solution is also returned by the algorithm. Note that this notion of a decision algorithm includes algorithms that may be incomplete, i.e., may fail to return a correct result within bounded time, or even incorrect, i.e., sometimes return erroneous results. In the following, we will focus on decision algorithms that are correct, but incomplete; this captures most heuristic decision algorithms, including, for example, almost all SLS algorithms for SAT.

14.2.1 Analysis on Single Instances

The primary performance metric for complete (and correct) decision algorithms is typically *run time*, i.e., the time required for solving a given problem instance. For incomplete algorithms, it may happen that, although the given problem instance is soluble, a solution cannot be found. (In this case, the algorithm may not terminate, or signal failure, for example, by returning 'no solution found'.) Obviously, such cases need to be noted; by further analysing them, valuable insights into weaknesses of the algorithm (or errors in its implementation) can be obtained.

Run time is typically measured in terms of CPU time (rather than wall-clock time) to minimise the impact of other processes that are running concurrently (e.g., system processes). Obviously, CPU time measurements are always based on a concrete implementation and run-time environment, i.e., machine and operating system; to facilitate reproducibility and comparability, a specification of the run-time environment (comprising at least the processor type and model, clock speed and amount of RAM, as well as the operating system, including version number) should be given along with any CPU time result.

It is often desirable to further abstract from details of the implementation and run-time environment, especially in the context of comparative performance studies. This can be achieved using *operation counts*, which reflect the number of elementary operations that are considered to contribute significantly towards an algorithm's performance, and *cost models*, which relate the cost of these operations (typically in terms of run time per execution) relative to each other or absolute in terms of CPU time for a given implementation and run-time environment [3]. For SLS algorithms, a commonly used operation count is the number of local search steps. When measuring performance in terms of operation counts, care should be taken to select elementary operations whose cost per step is constant or close to constant within and between runs of the algorithm on the same instance. In this situation, operation counts and CPU-time measurements are related to each other by scaling with a constant factor that only depends on the given problem instance. Using operation counts and an associated cost model rather than CPU-time measurements as the basis for empirical studies often gives a clearer and more detailed picture of algorithmic performance.

While performance analysis of deterministic decision algorithms on a single problem instance consists of a simple run-time measurement, matters are slightly more involved if the algorithm under consideration is randomised. In that case, the run time of an algorithm A applied to a given problem instance π corresponds to a random variable $RT_{A,\pi}$; the probability distribution of $RT_{A,\pi}$ is called the *run-time distribution* (*RTD*) *of A on π*. Clearly, the run-time behaviour of an algorithm A on a given problem instance π is completely and precisely characterised by the respective RTD. Furthermore, this RTD can be estimated based on run-time measurements obtained from multiple independent runs of A on π. For sufficiently high numbers of runs, the *empirical RTDs* thus obtained approximate the underlying theoretical RTD arbitrarily accurately. In practice, empirical RTDs based on 20–100 runs are sufficient for most purposes (this will be further discussed later in this chapter).

Graphical representations of empirical RTDs are often useful; plots of the respective cumulative distribution functions (CDFs) are easily obtained (see Ref. [1]) and, unlike histograms, show the underlying data in full detail. They also make it easy to read quantiles and quantile ratios (such as the median and quartile ratio) directly off the plots; these basic descriptive statistics provide the basis for quantitative analyses and many statistical tests, which are discussed later. Compared to averages and empirical standard deviations, medians and quantile ratios have the advantage of being less sensitive with respect to outliers. Given the fact that the RTDs of many randomised heuristic algorithms show very large variability, the stability of basic descriptive statistics can become an important consideration. For the same reason, empirical RTDs are often best presented in the form of semi-log or log-log plots. Figure 14.1 shows an example of a typical empirical RTD plot.

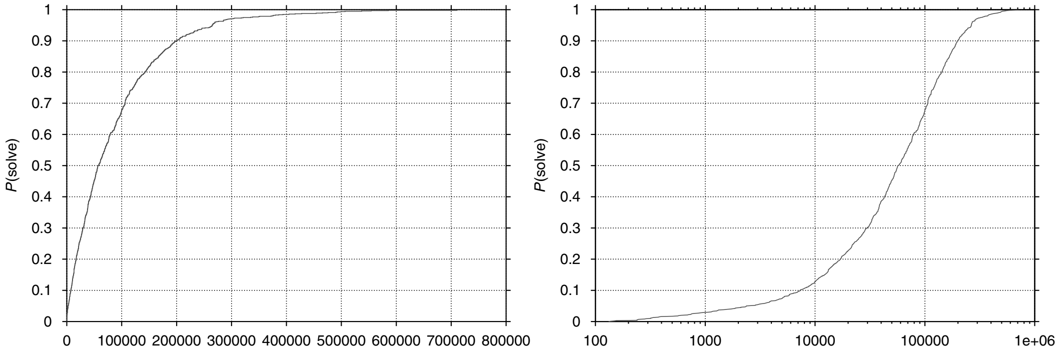


FIGURE 14.1 *Left*: Example of an empirical run-time distribution of an SLS algorithm for SAT applied to a hard problem instance; *right*: semi-log plot of the same RTD. P(solve) denotes the probability for finding a solution within the given run time.

14.2.2 Analysis on Instance Ensembles

Typically, the behaviour of heuristic algorithms is analysed on a set or ensemble of instances. The selection of such benchmark sets is an important factor in the design of an empirical study, and the use of inadequate benchmark sets can lead to questionable results and misleading conclusions. Although the criteria for benchmark selection depend significantly on the problem domain under consideration, on the hypotheses and goals of the empirical study and on the algorithms being analysed, there are some general principles and guidelines, which can be summarised as follows (for more details, see Ref. [1]): Benchmark sets should contain a diverse collection of problem instances, ideally including instances from real-world applications as well as artificially crafted and randomly generated instances; the instances should typically be intrinsically hard or difficult to solve for a broad range of algorithms. Furthermore, to facilitate the reproducibility of empirical analyses and the comparability of results between studies, it is important to use established benchmark sets (in particular those available from public benchmark libraries, such as ORLIB [4], TSPLIB [5] or SATLIB [6]), and to make newly created test-sets available to other researchers.

The basic approach to the empirical evaluation of an algorithm on a given ensemble of problem instances is to perform the same type of analysis described in the previous section on each individual instance. For small ensembles, it is often possible to analyse and report the results of this analysis for all instances, for example, in the form of tables or multiple RTD plots. When dealing with bigger ensembles, such as benchmark sets obtained from random instance generators, it becomes important to characterise the performance of a given algorithm on individual instances as well as across the entire ensemble. The latter can be achieved by aggregrating the results obtained on all individual instances into a so-called *search cost distribution* (*SCD*). For a deterministic algorithm applied to a given benchmark set, the empirical SCD is obtained from the run-time measurements on each individual problem instance. Analogous to RTDs, SCDs are typically best analysed qualitatively by means of CDF plots and quantitatively by means of basic descriptive statistics, such as quantiles and quantile ratios. For randomised decision algorithms, SCDs can be computed based on the median (or mean) run times for each individual instance; this means that each point in the SCD plot corresponds to a statistic of an entire RTD. It is often appropriate to also analyse in more detail a small set of RTDs that have been carefully selected in such a way that they representatively illustrate the variation in algorithm behaviour across the ensemble.

In many cases, it is also of considerable interest to investigate the dependence of algorithmic performance on certain instance features, such as problem size. This is often done by studying the correlation between the feature value for a given problem instance and the corresponding run time (or RTD) across the ensemble, for example, by means of simple correlation plots or using appropriate statistics, such as the Pearson correlation coefficient, and possibly also significance tests. The issues faced in this context are very similar to those arising in the comparative analysis of multiple algorithms on instance ensembles and will be further discussed in Section 14.2.4. In terms of qualitative analyses, choosing an appropriate graphical representation, such as a semilogarithmic plot for the functional dependence of mean cost on problem size, is often the key for easily detecting interesting behaviour (e.g., exponential scaling).

14.2.3 Comparative Analysis on Single Instances

In many empirical studies, the main goal is to establish the superiority of one heuristic algorithm over another. The most basic form of this type of analysis is the comparative analysis between two decision algorithms on a single problem instance. If both algorithms are deterministic, this amounts to a straightforward comparison between the respective run-time measurements. Clearly, in the case of incomplete algorithms or prematurely terminated runs, it needs to be noted if one or both algorithms failed to solve the given problem instance.

If at least one of the algorithms is randomised, the situation is slightly more complicated. Intuitively, an algorithm A shows superior performance compared to another algorithm B on a given problem instance π if for no run time, A has a lower solution probability than B, and there are some run times for which the solution probability of A is higher than that of B. In that case, we say that *A probabilistically dominates B*

TABLE 14.1 Upper Bounds on the Performance Differences Detectable by the Mann–Whitney U-Test for Various Sample Sizes (Number of Runs per RTD)

Significance Level 0.05, power 0.95		Significance Level 0.01, power 0.99	
Sample size	m_1/m_2	Sample size	m_1/m_2
3010	1.1	5565	1.1
1000	1.18	1000	1.24
122	1.5	225	1.5
100	1.6	100	1.8
32	2	58	2
10	3	10	3.9

Notes: m_1/m_2 denotes the ratio between the medians of the two given RTDs. The values in this table have been obtained using a standard procedure based on adjusting the statistical power of the two-sample t-test to the Mann–Whitney U-test using a worst-case Pitman asymptotic relative efficiency (ARE) value of 0.864.

on π (see Ref. [1]). A probabilistic domination relation holds between two decision algorithms on a given problem instance if, and only if, their respective cumulative RTD graphs do not cross each other. This provides a simple method for graphically checking probabilistic domination between two SLS algorithms on individual problem instances. The concept of probabilistic domination also applies to situations where one of A and B is deterministic, since in terms of analysing run-time behaviour, deterministic decision algorithms can be seen as special cases of randomised decision algorithms that have degenerate RTDs whose CDFs are simple step functions. In situations where a probabilistic domination relation does not hold, that is, there is a crossover between the respective RTD graphs, which of the two given algorithms is preferable in terms of higher-solution probability depends on the time the algorithms are allowed to run.

Statistical tests can be used to assess the significance of empirically measured performance differences between randomised algorithms. In particular, the *Mann–Whitney U-test* (or, equivalently, the *Wilcoxon rank sum test*) can be used to determine whether the medians of two given RTDs are equal [7]; a rejection of this null hypothesis indicates significant performance differences. The widely used t-test compares the means of two populations, but it requires the assumption that the given samples are normally distributed with identical variance—an assumption which is usually not met when analysing individual RTDs. The more specific hypothesis whether the theoretical RTDs of two decision algorithms are identical can be tested using the *Kolmogorov–Smirnov test for two independent samples* [7].

An important question arising in comparative performance analyses of randomised algorithms is that of sample size: How many independent runs should be performed when measuring the respective empirical RTDs? Generally, the ability of statistical tests to correctly distinguish situations in which the given null hypothesis is correct from those where it is incorrect crucially depends on sample size. This is illustrated in Table 14.1, which shows the performance differences between two given RTDs that can be detected by the Mann–Whitney U-test for standard significance levels and power values in dependence of sample size. (The significance level and power value indicate the maximum probabilities that the test incorrectly rejects or accepts the null hypothesis that the medians of the given RTDs are equal, respectively.)

In cases where probabilistic domination does not hold, the previously mentioned statistical tests are still applicable. However, they do not capture interesting and potentially important performance differences that can be easily seen from the respective RTD graphs. Such an example is depicted in Figure 14.2.

14.2.4 Comparative Analysis on Instance Ensembles

Comparative performance analyses of two decision algorithms on ensembles of problem instances are based on the same data used in the comparative analysis on the respective single instances. When dealing with two deterministic decision algorithms, A and B, this results in pairs of run times for each problem instance. In many cases, particularly when evaluating algorithms on big and diverse benchmark sets, there will be cases where A performs better than B and vice versa. In such situations it can be beneficial to use

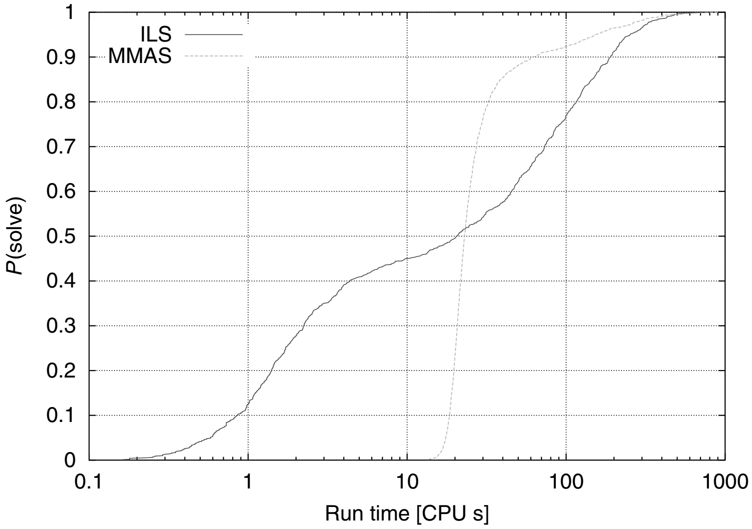


FIGURE 14.2 RTDs for two SLS algorithms for the TSP that for a given benchmark instance are required to find an optimal solution. Between 20 and 30 CPU s the two RTDs cross over.

statistical tests to assess the significance of the observed performance differences; this is particularly the case for benchmark sets obtained from random instance generators. The *binomial sign test* as well as the *Wilcoxon-matched pairs signed-rank test* determine whether the median of the paired differences is statistically significantly different from zero, indicating that one algorithm performs better than the other [7]. The Wilcoxon test is more sensitive, but requires the assumption that the distribution of the paired differences is symmetric. The well-known *t-test for two dependent samples* requires assumptions on the normality and homogeneity of variance of the underlying distributions of search cost over the given instance ensembles, which are typically not satisfied when dealing with the run times of heuristic algorithms.

If one or both of the given algorithms are randomised, the same tests can be applied to RTD statistics, such as the median (or mean) run time. However, this approach does not capture qualitative differences in performance, particularly as given in cases where there is no probabilistic domination of one algorithm over the other, and may suffer from inaccuracies due to a lack of statistical stability of the underlying RTD statistics. Therefore, additional analyses should be performed. In particular, the statistical significance of the performance differences (such as median run time) on each individual problem instance should be investigated using an appropriate test (such as the Mann–Whitney U-test). Furthermore, for each instance it should be checked whether a probabilistic domination relation holds; based on this information, the given instance ensemble can be partitioned into three subsets: (i) those instances on which A probabilistically dominates B, (ii) those on which B probabilistically dominates A, and (iii) those for which probabilistic domination is not observed. The relative sizes and contents of these partitions give a rather realistic and detailed picture of the algorithms' relative performance on the given set of instances.

Particularly for large instance ensembles, it is often useful to study the correlation between the performance of algorithms A and B across the given set of instances. This type of analysis can help to expose (and ultimately, remedy) weaknesses of an algorithm and to refine claims about its relative superiority for certain types of problem instances. For qualitative analyses of performance correlation, scatter plots can be used in which each instance is represented by one point, whose coordinates correspond to the performance of A and B applied to that instance. Performance measures used in this context are typically run time in case of deterministic algorithms, and RTD statistics, such as the median run time, otherwise. It should be noted that in the case of randomised algorithms, statistical instability of RTD statistics due to sampling error limits the accuracy of performance measurements. An example of such an analysis is shown in Figure 14.3.

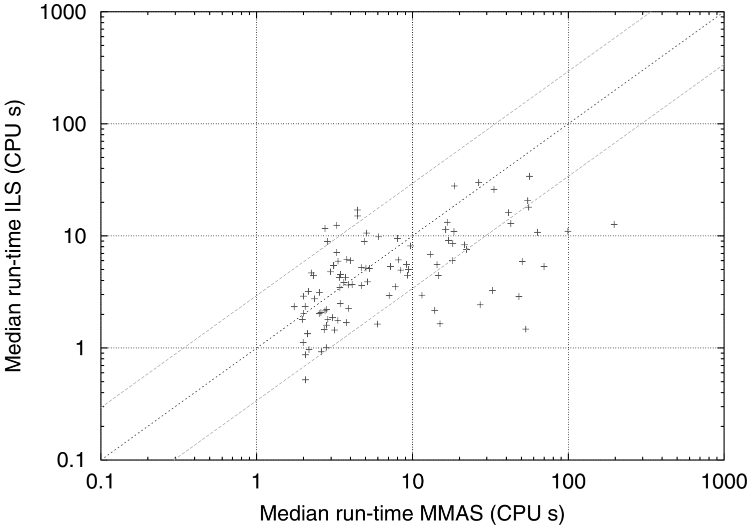


FIGURE 14.3 Correlation between the median run times required by two high-performance SLS algorithms for finding optimal solutions to a set of 100 TSP instances of 300 cities each; each median was measured across 10 runs per algorithm. The band between the two outer lines indicates performance differences that cannot be assumed to be statistically significant for the given sample size of the underlying RTDs.

Quantitatively, the correlation can be summarised using the empirical correlation coefficient. When the nature of an observed performance correlation seems to be regular (e.g., a roughly linear trend in the scatter plot), a simple regression analysis can be used to model the corresponding relationship in the algorithms' performance. It is often useful to perform correlation analyses on log-transformed data; this facilitates capturing general polynomial relationships.

To test the statistical significance of an observed performance correlation, nonparametric tests, such as *Spearman's rank order test* or *Kendall's tau test*, can be employed [7]. These tests determine whether there is a significant monotonic relationship in the performance data. They are preferable over tests based on Pearson's product-moment correlation coefficient, which require the assumption that the two random variables underlying the performance data stem from a bivariate normal distribution. (This assumption is often violated when dealing with run times of heuristic algorithms over instance ensembles.)

14.3 Optimisation Algorithms

In many situations, the objective of a computational problem is to find a solution that is optimal with respect to some measure of quality or cost. An example of such an *optimisation problem* is the widely studied TSP: given an edge-weigthed graph G, find a Hamiltonian cycle with minimal total weight, i.e., a shortest round trip that visits every vertex of G exactly once. Another example is MAX-SAT, the optimisation variant of the SAT problem, where the objective is to find an assignment of truth values to the propositional variables in a given formula F in conjunctive normal form such that a maximal number of clauses of F are simultaneously satisfied.

The measure to be optimised in an optimisation problem is called the *objective function*, and the term *solution quality* is used to refer to the objective function value of a given candidate solution. In most cases, solution qualities take the form of real numbers, and the goal is to find a candidate with either minimal or maximal solution quality. Optimisation problems can include additional logical conditions that any candidate solution needs to satisfy to be deemed *valid* or *feasible.* In the case of the TSP, such a logical condition states that to be considered a valid solution, a path in the given graph must be a Hamiltonian cycle. Logical conditions can always be integrated into the objective function in such a way that valid solutions are characterised by objective function values that exceed a specific threshold in solution quality.

An *optimisation algorithm* is an algorithm that takes as an input an instance of a given optimisation problem and returns a valid solution (or may determine that no valid solution exists). Optimisation algorithms that are theoretically guaranteed to find an optimal solution for any soluble problem instance within bounded time are called *complete* or *exact*; algorithms that are guaranteed to always return a solution that is within a specific constant factor of an optimal solution are called *approximation algorithms*.

When evaluating the performance of optimisation algorithms (theoretically or empirically), it is often useful to study the ratio between the solution quality achieved by the algorithm, q, and the optimal solution quality for the given problem instance, q^*. This performance measure is called the *approximation ratio*; formally, to be uniformly applicable to minimisation and maximisation problems, it is defined as $r := \max\{q/q^*, q^*/q\}$. When used in the empirical analysis of optimisation algorithms, solution qualities are often expressed in percent deviation from the optimum; this measure of *relative solution quality* is defined as $q' := (r - 1) \times 100$. For most heuristic optimisation algorithms, in particular for those based on SLS methods, there is a trade-off between run time and solution quality: the longer the algorithm is run, the better solutions are produced. The characterisation of this trade-off is of significant importance in the empirical analysis of optimisation algorithms.

14.3.1 Analysis on Single Instances

As in the case of decision algorithms, the empirical analysis of a deterministic optimisation algorithm on a single problem instance is rather straightforward, and many of the same considerations (particularly with respect to measuring run times and failure to produce valid solutions) apply. Run time / solution quality trade-offs are characterised by the development of solution quality over time, in the form of so-called *solution quality over time (SQT) curves*; these represent for each point in time t the quality of the best solution seen up to time t (the so-called *incumbent solution*) and are hence always monotone.

A slightly more complicated situation arises when dealing with randomised optimisation algorithms. Following the same approach as for randomised decision algorithms, run time is considered a random variable; in addition, a second random variable is used to capture solution quality, and the joint probability distribution of these two random variables characterises the behaviour of the algorithm on a given problem instance precisely and completely. For a given algorithm and problem instance, this probability distribution is called the *bivariate RTD of A on π* [1]; it can be visualised in the form of a cumulative distribution surface, each point of which represents the probability that A applied to π reaches (or exceeds) a certain solution quality bound within a certain amount of time (see Figure 14.4).

Empirical bivariate RTDs can be easily determined from multiple *solution quality traces*, each of which represents the development of solution quality over time for a single run of the algorithm on the given problem instance. A solution quality trace usually consists of pairs (t, q) for each point in time t at which an improvement in the incumbent solution, i.e., a new best solution quality q within the current run, has been achieved. As in the case of the (univariate) RTDs for decision algorithms, a sufficient number of independent runs (i.e., solution quality traces) on any given problem instance is required for measuring reasonably accurate empirical bivariate RTDs; obviously, the same holds for basic descriptive RTD statistics on the solution quality obtained within a given run time, or the run time required for reaching a given solution quality.

Multivariate probability distributions are more difficult to handle than univariate distributions. Therefore, rather than working directly with bivariate RTDs, it is often preferable to focus on the (univariate) distributions of the run time required for reaching a given solution quality threshold instead. These *qualified run-time distributions (QRTDs)* are the marginals of a given bivariate RTD for a specific bound on solution quality; intuitively, they correspond to cross-sections of the respective two-dimensional cumulative RTD graph for fixed solution quality values (see Figure 14.4). QRTDs directly characterise the ability of an SLS algorithm for an optimisation problem to solve the associated decision problems for the given solution quality bound. They are particularly useful for analysing an algorithm's ability to find optimal, close-to-optimal or feasible solutions and can be studied using exactly the same techniques as those

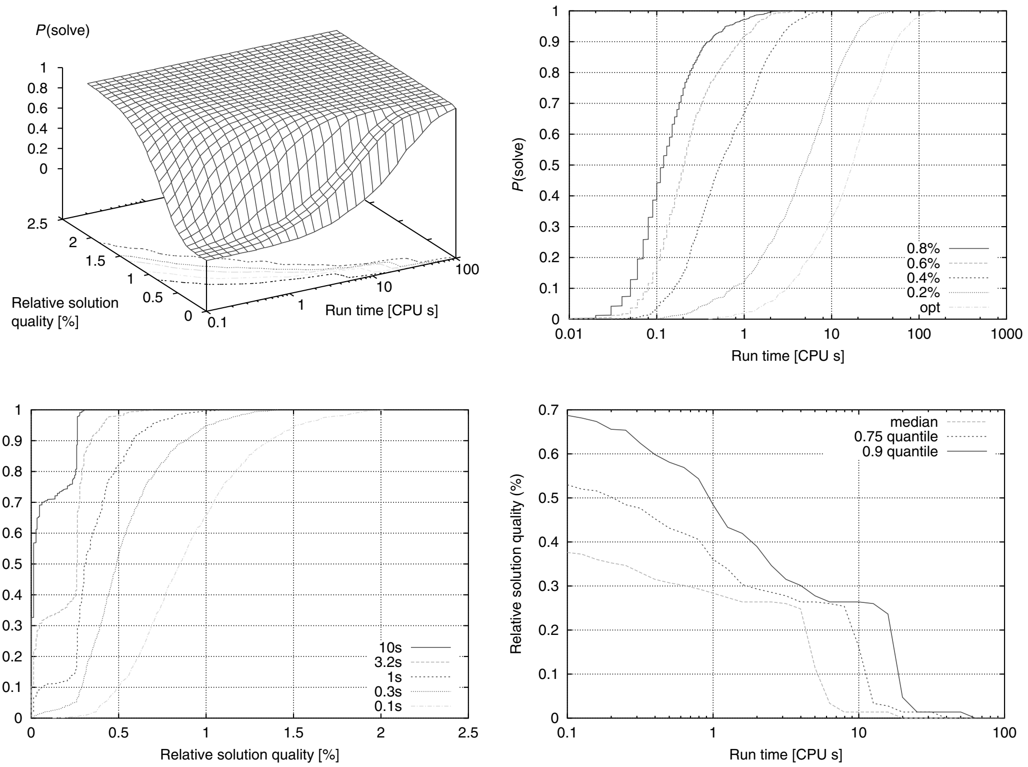


FIGURE 14.4 *Top left*: Bivariate RTD for an SLS algorithm applied to a TSP benchmark instance; the other plots give different views on the same distribution; *top right*: QRTDs for various relative solution quality bounds (percentage deviation from optimum); *bottom left*: SQDs for various run-time bounds (in CPU s); and *bottom right*: SQT curves for various SQD quantiles.

applied to the (univariate) RTDs of decision algorithms. A detailed picture of the behaviour of a randomised optimisation algorithm on a single problem instance can be obtained by analysing series of qualified RTDs for increasingly tight solution quality thresholds. The solution quality bounds used in a QRTD analysis are typically derived from knowledge of optimal solutions or bounds on the optimal solution quality; the latter case includes bounds obtained from long runs of heuristic optimisation algorithms.

Another commonly used way of studying the behaviour of randomised optimisation algorithms on a given problem instance is to analyse the distribution of solution qualities obtained over multiple independent runs with a fixed time bound. Technically, these so-called *solution quality distributions* (*SQDs*) are the marginals of the underlying bivariate RTDs for a fixed run-time bound. They correspond to cross-sections of the two-dimensional cumulative RTD graph for fixed run-time values; in this sense, they are orthogonal to QRTDs (see Figure 14.4). Again, these univariate distributions can be studied using essentially the same techniques as for analysing the RTDs of decision algorithms.

Closely related to SQDs are the *asymptotic solution quality distributions* obtained in the limit for arbitrarily long run times. For complete and probabilistically approximately complete optimisation algorithms, which are guaranteed to find an optimal solution to any given problem instance with arbitrarily high probability given sufficiently long run time, the asymptotic SQDs are degenerate distributions whose probability mass is completely concentrated on the optimal solution quality of the given problem instance. When dealing with randomised optimisation algorithms with an algorithm-dependent termination criterion, such as randomised iterative improvement methods that terminate upon reaching a local minimum, it is often also useful to study *termination time distributions (TTDs)*, which characterise the distribution of the time until termination over multiple independent runs.

Finally, the SQT curves described earlier in the context of characterising run time / solution quality trade-offs for deterministic optimisation algorithms can be generalised to randomised algorithms. This is done by replacing the uniquely defined solution quality values obtained by a deterministic algorithm for any given run-time bound by statistics of the respective SQDs in the randomised case. Although historically, this type of analysis has most commonly used SQT curves based on mean solution quality values, it is often preferable to use SQTs that reflect the development of SQD quantiles (such as the median) over time, since these tend to be statistically more stable than means. SQTs based on SQD quantiles also offer the advantage that they directly correspond to horizontal sections or contour lines of the underlying bivariate RTD surfaces. Combinations of such SQTs can be very useful for summarising certain aspects of a complete bivariate RTD; they are particularly well suited for analysing trade-offs between run time and solution quality (see Figure 14.4). However, the investigation of individual SQTs offers a fairly limited view of an optimisation algorithm's run-time behaviour in which important details can be easily missed and should therefore be complemented with other approaches, such as QRTD or SQD analysis. All these analyses can be carried out on the same set of solution quality traces collected over multiple indendent runs of the algorithm.

14.3.2 Comparative Analysis on Single Instances

The basic approach used for the comparative analysis of two (or more) optimisation algorithms on a single problem instance is analogous to that for decision algorithms. Often, a fixed target solution quality is used in this context, in which case the analysis involves the QRTDs of the algorithms with respect to that solution quality bound. Alternatively, a bound on run time can be used, and the respective SQDs can be compared using the same methods as in the case of RTDs for decision algorithms. (It may be noted that the SQDs of high-performance algorithms for high run times typically have much lower variance than QRTDs.)

Both of these methods do not take into account trade-offs between run time and solution quality. To capture such trade-offs, it is useful to extend the concept of domination introduced earlier for decision algorithms. We first note that in the case of two deterministic optimisation algorithms, A and B, this is straightforward: A dominates B on a given problem instance π if A gives consistently better solution quality than B for any run time. This implies that the respective SQT curves do not cross each other. In the case of crossing SQTs, which of the two algorithm is preferable in terms of solution quality achieved depends on the time the algorithms are allowed to run.

When generalised to randomised algorithms, this leads to the concept of *probabilistic domination*. Analogous to the case of randomised decision algorithms, probabilistic domination between two randomised optimisation algorithms holds if, and only if, their (bivariate) cumulative RTD surfaces do not cross each other. Note that this implies that there is no crossover between any SQDs for the same run-time bound, or between any QRTDs for any solution quality bound. In practice, probabilistic domination can be tested based on a series of QRTDs for different solution quality bounds (or SQDs for various run-time bounds). This does not require substantial experimental overhead, since the solution quality traces underlying empirical QRTDs for the best solution quality bound also contain all the information for QRTDs for lower-quality bounds. When probabilistic domination does not hold, the run-time/solution quality trade-offs between the given algorithms can be characterised using the same data. In many cases, the results from empirical performance comparisons between randomised optimisation algorithms can be conveniently summarised using SQT curves over multiple SQD statistics (e.g., median and additional quantiles) in combination with SQD plots for selected run times.

14.3.3 Analysis on Instance Ensembles

The considerations arising when extending the analyses described in the previous sections to ensembles of problem instances are essentially the same as in the case of decision algorithms (see Sections 14.2.2 and 14.2.4). It is convienent (and in some special cases sufficient) to perform the analysis for a single solution quality or run-time bound, in which case the methodology is analogous to that for decision algorithms. However, in most cases, run time / solution quality trade-offs need to be considered. This can be achieved by analysing SCDs or performance correlations for multiple solution quality or run-time bounds in addition to a more detailed analysis for carefully selected individual instances.

In the analysis of optimisation algorithms on instance ensembles, it is typically much preferable to use relative rather than absolute solution qualities. This introduces a slight complication when dealing with benchmark instances for which (provably) optimal solution qualities are unknown. To deal with such instances, theoretically or empirically determined bounds on the optimal solution quality, including best solution qualites achieved by high-performance heuristic algorithms, are often used. In this context, particularly when conducting performance comparisons related to the ability of various algorithms to find optimal or close-to-optimal solutions, it is very important to ensure that the bounds used in lieu of provably optimal solutions are as tight as possible.

14.4 Advanced RTD-Based Analysis

The measurement of RTDs for decision and optimisation problems can serve not only as a first step in the descriptive and comparative analysis of algorithm behaviour, as shown in the previous sections, but it can also form the basis of more advanced analysis techniques, for example, for examining scaling behaviour or performance robustness with respect to an algorithm's parameter settings. In what follows, we briefly outline such types of analyses; while our discussion is focused on RTDs for decision algorithms or, equivalently, on QRTDs for optimisation algorithms, many of its aspects can be extended in a straightforward way to the analysis of SQDs for optimisation algorithms.

14.4.1 Scaling with Instance Size

An important question is how an algorithm's performance scales with the size of the given problem instance. One approach to studying scaling behaviour is to base the analysis on individual instances of various sizes. However, since there is often very substantial variation in run time between instances of the same size, scaling studies are better based on ensembles of instances for each size. Then, the set of techniques discussed in the previous section can be applied by first measuring RTDs on individual instances; next, SCDs can be derived from appropriately chosen statistics of these RTDs, as discussed in Section 14.2.2; and finally, various statistics of these SCDs can be analysed in dependence of instance size.

As a first step, it is often useful to analyse the scaling data graphically. In this context, the use of semi-log or log-log plots can be very helpful: in particular, exponential scaling of mean or median search cost is reflected in a linear relationship between instance size and the logarithm of run time, while a linear relationship between the logarithms of both, instance size and run time is indicative of polynomial scaling. To analyse scaling behaviour in more detail, function fitting techniques, such as statistical regression, can be used. A simple example of an empirical scaling analysis is given in Figure 14.5.

Additional support for observed or conjectured scaling behaviour can be obtained by interpolation experiments, where for instance sizes that are in the range of the previously analysed instance ensembles additional data points are measured, or by extrapolation experiments, where an empirically fitted scaling function is used to predict the SCD statistics for larger instance sizes and deviations from the predicted values are analysed to possibly further refine the hypothesis on the scaling behaviour.

14.4.2 Impact of Parameter Settings

Many heuristic algorithms have one or more parameters that control their behaviour; as an example, consider the tabu tenure parameter in tabu search, a well-known SLS method (see also Chapters 19 and 23). The settings of such control parameters often have a significant yet theoretically poorly understood impact on the performance of the respective algorithm, which can be empirically studied by analysing the variation of an algorithm's RTD (or RTD statistics) in response to changes in its parameter settings. Often, the data required for this type of parameter sensitivity analysis is readily available from experiments conducted to optimise parameter settings for achieving peak performance.

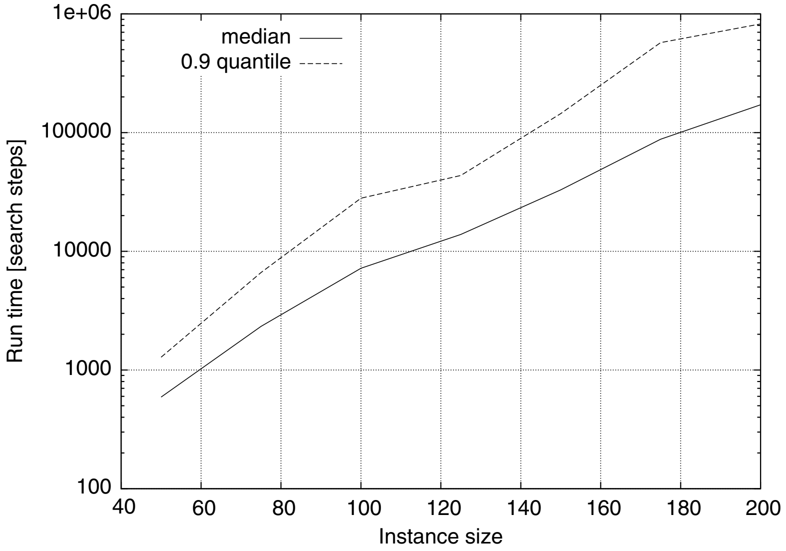


FIGURE 14.5 Scaling of the median and the 0.9 percentile for the search cost of solving SAT-encoded graph colouring instances with an SLS algorithm. Both statistics show evidence of exponential scaling.

It should be noted that in the case of randomised algorithms, the variation of run time for a fixed parameterisation and problem instance often depends on the parameter settings and should therefore be studied. For many SLS algorithms, suboptimal parameter values can cause search stagnation and extremely high variability in run time; in such situations, larger sample sizes may be required for obtaining reasonably accurate estimates of RTD statistics. Furthermore, for many heuristic algorithms with multiple parameters, the effects of various parameters are typically not independent, and experimental design techniques have to be employed for studying the nature and strength of these parameter dependencies.

Another important aspect of investigating parameter-dependent algorithmic performance deals with consistency across instance ensembles, i.e., with the question to which degree the impact of parameter settings is similar across the instances in a given ensemble. One way of approaching this issue is to treat different parameterisations like different algorithms, and to use the methods for comparative performance analysis on instance ensembles from Section 14.2.4 (in particular, correlation analysis of RTD statistics). Consistency of performance-optimising parameter settings is often of particular interest. When consistent behaviour across an ensemble is not observed, it may still be possible to relate aspects of parameter-dependent run-time behaviour to specific characteristics of the instances. Such characteristics could be of purely syntactic nature (such as instance size or clauses/variables ratio for SAT instances) or they may be based on some deeper semantic properties (such as search space features in the case of SLS algorithms).

The need for manually tuning parameters can cause problems in practical applications of heuristic algorithms as well as in their empirical analysis. In particular, comparative performance analyses can yield misleading results when parameter settings have been tuned unevenly (i.e., more effort has been spent in optimising parameter settings for one of the algorithms). To alleviate these problems, automatic tuning techniques have been proposed [8,9]. Furthermore, mechanisms for adapting parameter values while solving a given problem instance have been used with considerable success, in particular in the context of reactive search methods (see Chapter 21).

14.4.3 Stagnation Detection

Intuitively, a randomised heuristic decision algorithm shows stagnation behaviour if for long runs, the probability of finding a solution can be improved by restarting the algorithm at some appropriately chosen cut-off time. For search algorithms, this effect may be due to the inability of the algorithm to trade off

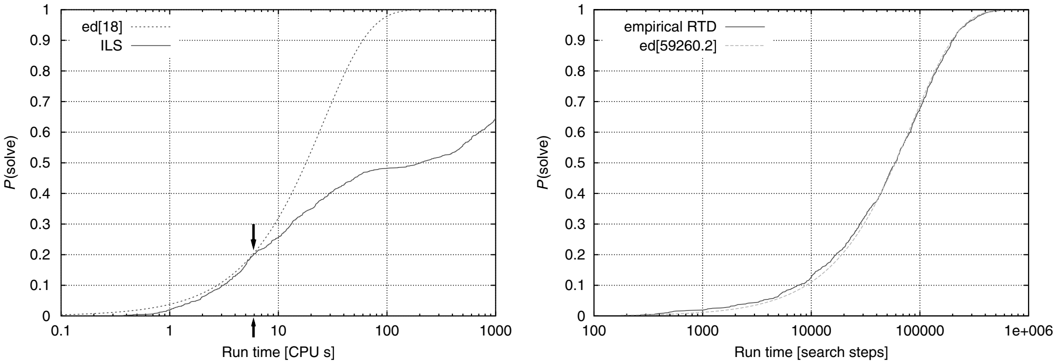


FIGURE 14.6 *Left*: Empirical QRTD of an iterated local search algorithm for finding the optimal solution of TSPLIB instance pcb442 (ILS); comparison with an exponential distribution ($\mathtt{ed}[m] = 1 - 2^{-\mathtt{run\,time}/m}$) reveals severe stagnation behaviour. *Right*: Best fit of an empirical RTD by an exponential distribution. The fit passes a χ^2 goodness-of-fit test at a significance level of $\alpha = 0.05$.

effectively exploration of the search space and exploitation of previous search experience, and may be related to the algorithm getting trapped in specific areas of the search space.

Interestingly, it is relatively straightforward to detect such stagnation behaviour from an empirical RTD. It is easy to see that only for RTDs that are identical to an exponential distribution, a well-known probability distribution from statistics, such restarts do not result in any performance loss or improvement [11] (essentially, this is due to the memory-less property of the exponential distribution). This insight provides the basis for detecting stagnation situations by comparing empirical RTDs of a given algorithm to exponential distributions. Stagnation behaviour is present if there is an exponential distribution whose CDF graph meets that of the empirical RTD from below but never crosses it. This situation is illustrated in Figure 14.6 (left pane); the arrows indicate the optimal cut-off time for a static restart strategy, which can also be determined from the RTD.

In general, the detection of stagnation situations using the RTD-based methodology can be a key element in the systematic development of randomised heuristic algorithms; for example, in the case of SLS algorithms, the occurrence of search stagnation often indicates the need for additional or stronger diversification mechanisms. (For further details, see Chapter 4 of Ref. [1].)

14.4.4 Functional Characterisation of Empirical RTDs

It is often useful (though not always possible) to characterise empirical RTDs by means of simple mathematical functions. For example, the RTDs of many high-performance SLS algorithms are well approximated by exponential distributions (see, e.g., Ref. [11]). Such characterisations are not only useful in the context of stagnation analysis (as explained in the previous section), but also provide detailed and often very accurate summarisations of an algorithm's run-time behaviour. Furthermore, they can help in gaining insights into an algorithm's properties by providing a basis for modelling its behaviour mathematically.

In the context of functional RTD characterisations, it is particularly appealing to model empirical RTDs using parameterised continuous probability distributions known from statistics. This can be done using standard fitting techniques to determine suitable parameter values; the quality of the resulting approximations can be evaluated using goodness-of-fit tests, such as the χ^2 test or the Kolmogorov–Smirnov test [7]. (For an illustration, see right pane of Figure 14.6.) The same methods can be used for functionally characterising other empirical data, such as SQDs or SCDs.

When dealing with large instance ensembles, the fitting and testing process needs to be automated. This way, more general hypotheses regarding an algorithm's run-time behaviour can be investigated empirically. Like any empirical approach, this method cannot be used for proving universal results on an algorithms' behaviour on an entire (infinite) class of problem instances, but it can be very useful in formulating, refining or falsifying hypotheses on such results.

14.5 Extensions

Most empirical analyses of heuristic algorithms in the literature focus on "classical" $\mathcal{NP}$-hard problems. It is clear, however, that sound empirical methodologies are equally important when tackling conceptually more involved types of problems, such as multiobjective, dynamic or stochastic optimisation problems.

Multiobjective problems. In multiobjective problems, several, typically conflicting optimisation criteria need to be considered simultaneously. For these problems, a common goal is to identify the set of *Pareto-optimal solutions* [12], i.e., solutions for which there exists no alternative that is strictly better with respect to all optimisation criteria. Such multiobjective problems arise in many engineering and business applications, and heuristic algorithms are widely used for solving them [13,14]. The behaviour of these algorithms can be analysed empirically using a suitably generalised notion of multivariate RTDs. Since the dimensionality of the RTDs to be measured in this case is equal to the number of objective functions plus one, data collection and analysis are considerably more complex than in the case of single-objective optimisation problems. While we are not aware of any studies based on these multivariate RTDs, the marginal distributions obtained when keeping the computation time fixed have received considerable attention. The analysis of these so-called *attainment functions* has been proposed by Fonseca et al. [15] and has been acknowledged as one of the few approaches for a correct analysis of the performance of randomised algorithms for multiobjective optimisation [16].

Dynamic problems. In many applications, some aspects of a given problem instance may change while trying to find or implement a solution. Such dynamic problems are encountered, for example, in many distribution problems, where traffic situations can change as a result of congested or blocked routes. Two common goals in dynamic problems are to minimise the delay in recovering solutions (of a certain quality) after a change in the problem instance has occurred and to miminise disruptions of the current solution, i.e., the amount of modifications required to adapt the current solution to the changed situation. The empirical analysis of heuristic (and in particular, randomised) algorithms for both of these situations can be handled using relatively straightforward extensions of the RTD-based methodology. In the case of dynamic optimisation problems, tradeoffs between solution quality and the amount of disruption can be studied using the same techniques as for static multiobjective problems. Also, particularly for dynamic optimisation problems where changes occur rather frequently, it can be useful to analyse the development of solution quality (or, for randomised algorithms, SQDs) over time, using suitable generalisations of the RTD-based techniques for static optimisation problems.

Stochastic problems. In some practical applications, important properties of solutions are subject to statistical variation. For many stochastic optimisation problems, variations in the quality of a given solution are caused by random changes (or uncertainty) in solution components that are characterised in the form of probability distributions; for example, in stochastic routing problems, the costs associated with using certain connections may be specified by Gaussian distributions. A typical goal when solving stochastic optimisation problems is to find a solution with optimal *expected quality*. In some cases, the expected quality of a solution can be determined analytically, and algorithms for such problems can be analysed using the same empirical methods as described for conventional deterministic problems. In other cases, approximation or sampling methods have to be used for estimating the quality of candidate solutions. While in principle, the techniques described in this chapter can be extended to these cases, empirical analyses (as well as algorithm development) are more involved; for example, when measuring empirical SQDs, a trade-off arises between the number of algorithm runs and the number of samples used to estimate the quality of incumbent solutions.

14.6 Further Reading

The use of principled and advanced techniques for the empirical analysis of deterministic and randomised heuristic algorithms is gaining increasing acceptance among researchers and practitioners. In this chapter, we have described the analysis of RTDs as a core technique for the empirical investigation and characterisation of randomised algorithms [17]. While RTDs have been previously reported in the literature [18–20], they have typically been used for purely descriptive purposes or in the context of investigating the parallelisation speed-up achievable by performing multiple independent runs of a sequential algorithm. A more detailed description of the RTD-based methodology is given in Chapter 4 of Ref. [1]. RTD-based methods are now being used increasingly widely for the empirical study of a broad range of SLS algorithms for numerous combinatorial problems [21–29].

SQDs of randomised heuristic optimisation algorithms have been occasionally reported in the literature; they have been used, for example, to obtain results on the scaling of SLS behaviour [30]. SQDs can also be used for estimating optimal solution qualities for combinatorial optimisation problems [31,32]. SCDs over ensembles of problem instances have been measured and characterised for deterministic, complete algorithms for binary constraint satisfaction problems and SAT [33,34].

There is a growing body of work on general issues in empirical algorithmics. Several articles provide guidelines for the experimental study of mathematical programming software [35,36] and heuristic algorithms [37], with the aim of increasing the reproducibility of results. General guidelines for the experimental analysis of algorithms have also been proposed by McGeoch and Moret [38–40]. Johnson [41] gives an overview of guidelines and potential pitfalls in empirical algorithmics research. A more scientific approach to experimental studies of algorithms in optimisation has been advocated by Hooker [42,43], who emphasised the need for formulating and empirically investigating hypotheses about algorithm properties and behaviour rather than limiting the experimental study of algorithms to simplistic performance comparisons.

At the core of any empirical approach to investigating the behaviour and performance of randomised algorithms are statistical methods. Cohen's book [44] provides a good introduction to empirical methods in computing science with an emphasis on algorithms and applications in artificial intelligence. The handbook by Sheskin [7] is an excellent source for detailed information on statistical tests and their application, while Siegel et al. [45] and Conover [46] provide more specialised introductions to nonparametric statistics. For an introduction to the important topic of experimental design and data analysis we refer to the books of Dean and Voss [47] and Montgomery [48].

Acknowledgments

HH acknowledges support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Discovery Grant 238788-05; TS acknowledges support received from the Belgian Fonds National de la Recherche Scientifique, of which he is a research associate.

References

[1] Hoos, H. H. and Stützle, T., *Stochastic Local Search—Foundations and Applications*, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.

[2] Hajek, B., Cooling schedules for optimal annealing, *Math. Oper. Res.*, 13(2), 311, 1988.

[3] Ahuja, R. K. and Orlin, J. B., Use of representative operation counts in computational testing of algorithms, *INFORMS J. Comput.*, 8(3), 318, 1996.

[4] Beasley, J. E., OR-Library, `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`, last visited February 2006.

[5] Reinelt, G., TSPLIB, `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95`, last visited February 2006.

[6] Hoos, H. H. and Stützle, T., SATLIB—The Satisfiability Library, `http://www.satlib.org`, last visited February 2006.

[7] Sheskin, D. J., *Handbook of Parametric and Nonparametric Statistical Procedures*, 2nd ed. Chapman & Hall / CRC, Boca Raton, FL, USA, 2000.

[8] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K., A racing algorithm for configuring metaheuristics, *Proc. Genetic and Evolutionary Computation Conf.*, 2002, p. 11.

[9] Coy, S. P., Golden, B. L., Runger, G. C., and Wasil, E. A., Using experimental design to find effective parameter settings for heuristics, *J. Heuristics*, 7(1), 77, 2001.

[10] Battiti, R., Reactive search: toward self-tuning heuristics, in *Modern Heuristic Search Methods*, Rayward-Smith, V. J., Ed., Wiley, New York, 1996, p. 61.

[11] Hoos, H. H. and Stützle, T., Characterising the behaviour of stochastic local search, *Artif. Intell.*, 112(1–2), 213, 1999.

[12] Steuer, R. E., *Multiple Criteria Optimization: Theory, Computation and Application*, Wiley, New York, USA, 1986.

[13] Gandibleux, X., Sevaux, M., Sörensen, K., and T'kindt, V., Eds., *Metaheuristics for Multiobjective Optimisation*, Lecture Notes in Economics and Mathematical Systems, Vol. 535, Springer, Berlin, 2004.

[14] Coello, C. A. and Lamont, G. B., Eds, *Applications of Multi-Objective Evolutionary Algorithms*, World Scientific, Singapore, 2004.

[15] Grunert da Fonseca, V., Fonseca, C. M., and Hall, A., Inferential performance assessment of stochastic optimizers and the attainment function, in *Evolutionary Multi-Criterion Optimization*, Zitzler, E. et al., Eds., Lecture Notes in Computer Science, Vol. 1993, Springer, Berlin, 2001, p. 213.

[16] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and Grunert da Fonseca, V., Performance assessment of multiobjective optimizers: an analysis and review, *IEEE Trans. Evolut. Comput.*, 7(2), 117, 2003.

[17] Hoos, H. H. and Stützle, T., Evaluating Las Vegas algorithms—pitfalls and remedies, *Proc. 14th Conf. on Uncertainty in Artificial Intelligence*, 1998, p. 238.

[18] Battiti, R. and Tecchiolli, G., Parallel biased search for combinatorial optimization: genetic algorithms and TABU, *Microprocess. Microsys.*, 16(7), 351, 1992.

[19] Taillard, É. D., Robust taboo search for the quadratic assignment problem, *Parallel Comput.*, 17(4–5), 443, 1991.

[20] ten Eikelder, H. M. M., Verhoeven, M. G. A., Vossen, T. V. M., and Aarts, E. H. L., A probabilistic analysis of local search, in *Metaheuristics: Theory & Applications*, Osman, I. H. and Kelly, J. P., Eds., Kluwer Academic Publishers, Boston, MA, 1996, p. 605.

[21] Aiex, R. M., Resende, M. G. C., and Ribeiro, C. C., Probability distribution of solution time in GRASP: an experimental investigation, *J. Heuristics*, 8(3), 343, 2002.

[22] Aiex, R. M., Pardalos, P. M., Resende, M. G. C., and Toraldo, G., GRASP with path relinking for three-index assignment, *INFORMS J. Comput.*, 17, 224, 2005.

[23] Braziunas, D. and Boutilier, C., Stochastic local search for POMDP controllers, *Proc. National Conf. on Artificial Intelligence*, 2004, p. 690.

[24] Hoos, H. H. and Boutilier, C., Solving combinatorial auctions using stochastic local search, *Proc. National Conf. on Artificial Intelligence*, 2000, p. 22.

[25] Hoos, H. H. and Stützle, T., Local search algorithms for SAT: an empirical evaluation, *J. Automated Reasoning*, 24(4), 421, 2000.

[26] Shmygelska, A. and Hoos, H. H., An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem, *BMC Bioinform.*, 6(30), 2005.

[27] Stützle, T. and Hoos, H. H., Analysing the run-time behaviour of iterated local search for the travelling salesman problem, in *Essays and Surveys on Metaheuristics*, Hansen, P. and Ribeiro, C. C., Eds., Kluwer Academic Publishers, Boston, MA, 2001, p. 589.

[28] Stützle, T., Iterated local search for the quadratic assignment problem, *Eur. J. Oper. Res.*, 174(3), 1519, 2006.

[29] Watson, J.-P., Whitley, L. D., and Howe, A. E., Linking search space structure, run-time dynamics, and problem difficulty: a step towards demystifying tabu search, *J. Artif. Intell. Res.*, 24, 221, 2005.

[30] Schreiber, G. R. and Martin, O. C., Cut size statistics of graph bisection heuristics, *SIAM J. Opt.*, 10(1), 231, 1999.

[31] Dannenbring, D. G., Procedures for estimating optimal solution values for large combinatorial problems, *Management Sci.*, 23(12), 1273, 1977.

[32] Golden, B. L. and Steward, W., Empirical analysis of heuristics, in *The Traveling Salesman Problem*, Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B., Eds., John Wiley & Sons, Chichester, UK, 1985, p. 207.

[33] Kwan, A. C. M., Validity of normality assumption in CSP research, in *PRICAI: Topics in Artificial Intell.*, Foo, N. Y. and Goebel, R., Eds., Lecture Notes in Computer Science, Vol. 1114, Springer, Berlin, 1996, p. 253.

[34] Frost, D., Rish, I., and Vila, L., Summarizing CSP hardness with continuous probability distributions, in *Proc. Nat. Conf. on Artificial Intelligence*, 1997, p. 327.

[35] Crowder, H., Dembo, R., and Mulvey, J., On reporting computational experiments with mathematical software, *ACM Trans. Math. Software*, 5(2), 193, 1979.

[36] Jackson, R., Boggs, P., Nash, S., and Powell, S., Report of the ad-hoc committee to revise the guidelines for reporting computational experiments in mathematical programming, *Math. Prog.*, 49, 413, 1990.

[37] Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C., and Stewart, W. R., Designing and reporting on computational experiments with heuristic methods, *J. Heuristics*, 1(1), 9, 1995.

[38] McGeoch, C. C., Toward an experimental method for algorithm simulation, *INFORMS J. Comput.*, 8(1), 1, 1996.

[39] McGeoch, C. C. and Moret, B. M. E., How to present a paper on experimental work with algorithms, *SIGACT News*, 30(4), 85, 1999.

[40] Moret, B. M. E., Towards a discipline of experimental algorithmics, in *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, Goldwasser, M. H., Johnson, D. S., and McGeoch, C. C., Eds., AMS, Providence, RI 2002, p. 197.

[41] Johnson, D. S., A theoretician's guide to the experimental analysis of algorithms, in *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, Goldwasser, M. H., Johnson, D. S., and McGeoch, C. C., Eds., AMS, Providence, RI 2002, p. 215.

[42] Hooker, J. N., Needed: an empirical science of algorithms, *Oper. Res.*, 42(2), 201, 1994.

[43] Hooker, J. N., Testing heuristics: we have it all wrong, *J. Heuristics*, 1(1), 33, 1996.

[44] Cohen, P. R., *Empirical Methods for Artificial Intelligence*, MIT Press, Cambridge, MA, USA, 1995.

[45] Siegel, S., Castellan Jr., N. J., and Castellan, N. J., *Nonparametric Statistics for the Behavioral Sciences*, 2nd ed., McGraw-Hill, New York, 2000.

[46] Conover, W. J., *Practical Nonparametric Statistics*, 3rd ed., Wiley, New York, USA, 1999.

[47] Dean, A. and Voss, D., *Design and Analysis of Experiments*, Springer, Berlin, Germany, 2000.

[48] Montgomery, D. C., *Design and Analysis of Experiments*, 5th ed., Wiley, New York, USA, 2000.

15

Reductions That Preserve Approximability

Giorgio Ausiello
University of Rome
"La Sapienza"

Vangelis Th. Paschos
LAMSADE CNRS UMR 7024 and
University of Paris–Dauphine

15.1 Introduction

The technique of transforming a problem into another in such a way that the solution of the latter entails, somehow, the solution of the former is a classical mathematical technique that has found wide application in computer science since the seminal works of Cook [1] and Karp [2], who introduced particular kinds of transformations (called *reductions*) with the aim of studying the computational complexity of combinatorial decision problems. The interesting aspect of a reduction between two problems consists in its twofold application: on the one hand it allows to transfer positive results (resolution techniques) from one problem to the other and, on the other hand, it may also be used for deriving negative (hardness) results. In fact, as a consequence of such seminal work, by making use of a specific kind of reduction, the polynomial-time *Karp-reducibility*, it has been possible to establish a complexity partial order among decision problems, which, for example, allows us to state that, modulo polynomial-time transformations, the SATISFIABILITY problem is as hard as thousands of other combinatorial decision problems, even though the precise complexity level of all these problems is still unknown.

Strictly associated with the notion of reducibility is the notion of completeness. Problems that are complete in a complexity class via a given reducibility are, in a sense, the hardest problems of such class. Besides, given two complexity classes **C** and $\mathbf{C}' \subseteq \mathbf{C}$, if a problem Π is complete in **C** via reductions that belong (preserve membership) to $\mathbf{C}'$, to establish whether $\mathbf{C}' \subset \mathbf{C}$, it is "enough" to assess the actual complexity of Π (informally, we say that Π is a candidate to separate **C** and $\mathbf{C}'$).

In this chapter we will show that an important role is played by reductions also in the field of approximation of hard combinatorial optimization problems. In this context, the kind of reductions which will be applied are called *approximation preserving reductions*. Intuitively, in the most simple case, an approximation preserving reduction consists of two mappings f and g: f maps an instance x of problem Π into an instance $f(x)$ of problem Π', g maps back a feasible solution y of Π' into a feasible solution $g(y)$ of Π with the property that $g(y)$ is an approximate solution of problem Π whose quality is almost as good as the quality of the solution y for problem Π'. Clearly, again in this case, the role of an approximation preserving reduction is twofold: on the one hand it allows to transfer an approximation algorithm from problem Π' to problem Π; on the other, if we know that problem Π cannot be approximated beyond a given threshold, such limitation applies also to problem Π'.

Various kinds of approximation-preserving reducibilities will be introduced in this chapter and we will show how they can be exploited in a positive way, to transform solution heuristics from a problem to another and how, on the contrary, they may help in proving negative, inapproximability results.

It is well known that **NP**-hard combinatorial optimization problems behave in a very different way with respect to approximability and can be classified accordingly. While for some problems there exist polynomial-time approximation algorithms that provide solutions with a constant approximation ratio w.r.t. the optimum solution, for some other problems even a remotely approximate solution is computationally hard to achieve. Analogous to what happens in the case of the complexity of decision problems, approximation-preserving reductions allow to establish a partial order among optimization problems in terms of approximability properties, independently from the actual level of approximation that for such problems can be achieved (and that in some cases is still undefined). Approximation-preserving reductions can also be used to define complete problems which play an important role in the study of possible separations between approximation classes. The discovery that a problem is complete in a given approximation class provides a useful insight in understanding what makes a problem not only computationally hard but also resilient to approximate solutions.

As a final remark on the importance of approximation-preserving reductions, let us observe that such reductions require some correspondence between the combinatorial structure of two problems be established. This is not the case for reductions between decision problems. For example, in such a case, we see that all **NP**-complete decision problems turn out to be mutually interreducible by means of polynomial-time reduction while when we consider the corresponding optimization problems, the different approximability properties come to evidence. As a consequence, we can say that approximation-preserving reductions are also a useful tool to analyze the deep relation existing between combinatorial structure of problems and the hardness of approximation.

The rest of this chapter is organized as follows. The next section is devoted to basic definitions and preliminary results concerning reductions among combinatorial optimization problems. In Section 15.3 we provide the first, simple example of approximation-preserving reducibility, namely the *linear reducibility*, that while not as powerful as the reducibilities that will be presented in the sequel is widely used in practice. In Section 15.4, we introduce the reducibility that, historically, has been the first to be introduced, *the strict reducibility* and we discuss the first completeness results based on reductions of such kind. Next, in Section 15.5, we introduce AP-reducibility, and in Section 15.6 we discuss more extensive completeness results in approximation classes. In Section 15.7, we present a new reducibility, called FT-reducibility, that allows to prove the polynomial-time approximation scheme (**PTAS**)-completeness of natural NP-optimization (**NPO**) problems. Finally, in Section 15.8, we present other reductions with the specific aim of proving further inapproximability results. The last two sections of the chapter contain conclusions and references.

In this chapter we assume that the reader is familiar with the basic notions of computational complexity regarding both decision problems and combinatorial optimization problems, as they are defined in Chapter 1.

15.2 Basic Definitions

Before introducing the first examples of reductions between optimization problems, let us recall the definitions of the basic notions of approximation theory and of the most important classes of optimization problems, characterized in terms of their approximability properties. First of all we introduce the class **NPO** which is the equivalent, for optimization problems, of the class of decision problems **NP**.

Definition 15.1

An **NP** optimization problem, **NPO**, Π *is defined as a four-tuple* $(\mathcal{I}, Sol, m, \text{goal})$ *such that*

- $\mathcal{I}$ *is the set of* instances *of* Π *and it can be recognized in polynomial time;*
- *given* $x \in \mathcal{I}$, $Sol(x)$ *denotes the set of* feasible solutions *of* x*; for any* $y \in Sol(x)$, $|y|$ *(the size of* y*) is polynomial in* $|x|$ *(the size of* x*); given any* x *and any* y *polynomial in* $|x|$, *one can decide in polynomial time if* $y \in Sol(x)$*;*
- *given* $x \in \mathcal{I}$ *and* $y \in Sol(x)$, $m(x, y)$ *denotes the value of* y *and can be computed in polynomial time;*
- goal $\in \{min, max\}$ *indicates the* type *of optimization problem.*

Given an **NPO** problem $\Pi = (\mathcal{I}, \text{Sol}, m, \text{goal})$ an optimum solution of an instance x of Π is usually denoted $y^*(x)$ and its measure $m(x, y^*(x))$ is denoted by $\text{opt}(x)$.

Definition 15.2

Given an **NPO** *problem* $\Pi = (\mathcal{I}, Sol, m, \text{goal})$, *an* approximation algorithm A *is an algorithm that given an instance* x *of* Π *returns a feasible solution* $y \in Sol(x)$. *If* A *runs in polynomial time with respect to* $|x|$, A *is called a* polynomial-time approximation algorithm *for* Π.

The quality of the solution given by an approximation algorithm $\mathtt{A}$ for a given instance x is usually measured as the ratio $\rho_A(x)$, *approximation ratio*, between the value of the approximate solution, $m(x, A(x))$, and the value of the optimum solution $opt(x)$. For minimization problems, therefore, the approximation ratio is in $[1, \infty)$, while for maximization problems it is in $[0, 1]$.

Definition 15.3

An **NPO** *problem* Π *belongs to the class* **APX** *if there exist a polynomial-time approximation algorithm* A *and a value* $r \in \mathbb{Q}$ *such that, given any instance* x *of* Π, $\rho_{\mathtt{A}}(x) \leq r$ *(resp.,* $\rho_{\mathtt{A}}(x) \geq r$*) if* Π *is a minimization problem (resp., a maximization problem). In such a case,* A *is called an* r*-approximation algorithm.*

Examples of combinatorial optimization problems belonging to the class **APX** are MAX SATISFIABILITY, MIN VERTEX COVER, and MIN EUCLIDEAN TSP.

In some cases, a stronger form of approximability for **NPO** problems can be obtained by a PTAS that is a family of algorithms $\mathtt{A}_r$ such that, given any ratio $r \in \mathbb{Q}$, the algorithm $\mathtt{A}_r$ is an r-approximation algorithm whose running time is bounded by a suitable polynomial p as a function of $|x|$.

Definition 15.4

An **NPO** *problem* Π *belongs to the class* **PTAS** *if there exists a* **PTAS** A_r *such that, given any* $r \in \mathbb{Q}$, $r \neq 1$, *and any instance* x *of* Π, $\rho_{\mathtt{A}_r}(x) \leq r$ *(resp.,* $\rho_{\mathtt{A}_r}(x) \geq r$*) if* Π *is a minimization problem (resp., a maximization problem).*

Among the problems in **APX** listed above, the problem MIN EUCLIDEAN TSP can be approximated by means of a **PTAS** and hence belongs to the class **PTAS**. Moreover, other examples of combinatorial optimization problems belonging to the class **PTAS** are MIN PARTITIONING and MAX INDEPENDENT SET ON PLANAR GRAPHS.

Finally, a stronger form of approximation scheme can be used for particular problems in **PTAS**, such as, for example, MAX KNAPSACK or MIN KNAPSACK. In such cases, in fact, the running time of the algorithm A_r is uniformly polynomial in r as made precise in the following definition.

Definition 15.5

An **NPO** *problem* Π *belongs to the class fully polynomial-time approximation scheme* **(FPTAS)** *if there exists a* **PTAS** A_r *such that, given any* $r \in \mathbb{Q}$, $r \neq 1$, *and any instance* x *of* Π, $\rho_{\mathtt{A}_r}(x) \leq r$ *(resp.,* $\rho_{\mathtt{A}_r}(x) \geq r$*) if* Π *is a minimization problem (resp., a maximization problem) and, furthermore, there exists a two variate polynomial* q *such that the running time of* $A_r(x)$ *is bounded by* $q(x, 1/(r-1))$ *(resp.,* $q(x, 1/(1-r))$ *in case of maximization problems).*

It is worth remembering that under the hypothesis that **P** $\neq$ **NP** all the above classes form a strict hierarchy that is **FPTAS** $\subset$ **PTAS** $\subset$ **APX** $\subset$ **NPO**.

Let us note that there also exists other notorious approximability classes, as **Poly-APX**, **Log-APX**, **Exp-APX**, the classes of problems approximable within ratios that are, respectively, polynomials (or inverse of polynomials if goal = max), logarithms (or inverse of logarithms), exponentials (or inverse of exponentials) of the size of the input. The best studied among them is the class **Poly-APX**. Despite their interest for sake of conciseness these classes are not dealt in the chapter.

When the problem of characterizing approximation algorithms for hard optimization problems was tackled, soon the need arose for a suitable notion of reduction that could be applied to optimization problems to study their approximability properties [3]:

> What is it that makes algorithms for different problems behave in the same way? Is there some stronger kind of reducibility than the simple polynomial reducibility that will explain these results, or are they due to some structural similarity between the problems as we define them?

Approximation preserving reductions provide an answer to the above question. Such reductions have an important role when we wish to assess the approximability properties of an **NPO** optimization problem and locate its position in the approximation hierarchy. In such a case, in fact, if we can establish a relationship between the given problem and other known optimization problems, we can derive both positive information on the existence of approximation algorithms (or approximation schemes) for the new problem or, on the other side, negative information, showing intrinsic limitations to approximability. With respect to reductions between decision problems, reductions between optimization problems have to be more elaborate. Such reductions, in fact, have to map both instances and solutions of the two problems, and they have to preserve, so to say, the optimization structure of the two problems.

The first examples of reducibility among optimization problems were introduced by Ausiello et al. in Refs. [4,5] and by Paz and Moran in Ref. [6]. In particular, in Ref. [5], the notion of *structure preserving reducibility* is introduced and for the first time the completeness of MAX WSAT (weighted-vertex SAT) in the class of **NPO** problems is proved. Still it took a few more years until suitable notions of approximation preserving reducibilities were introduced by Orponen and Mannila in Ref. [7]. In particular, their paper presented the strict reduction (see Section 15.4) and provided the first examples of natural problems who are complete under approximation preserving reductions: (MIN WSAT, MIN 0-1 LINEAR PROGRAMMING, and MIN TSP).

Before introducing specific examples of approximation preserving reduction in the next sections, let us explain more formally how reductions between optimization problems can be defined, starting from the notion of *basic reducibility* (called R-reducibility in the following, denoted $\leq_{\mathsf{R}}$) which underlays most of the reducibilities that will be later introduced.

Definition 15.6

Let Π_1 and Π_2 be two **NPO** *maximization problems. Then we say that $\Pi_1 \leq_{\mathsf{R}} \Pi_2$ if there exist two polynomial-time computable functions f, g that satisfy the following properties:*

- $f : \mathcal{I}_{\Pi_1} \to \mathcal{I}_{\Pi_2}$ *such that* $\forall x_1 \in \mathcal{I}_{\Pi_1}$, $f(x_1) \in \mathcal{I}_{\Pi_2}$; *in other words, given an instance x_1 in Π_1, f allows to build an instance $x_2 = f(x_1)$ in Π_2;*
- $g : \mathcal{I}_{\Pi_1} \times \mathrm{Sol}_{\Pi_2} \to \mathrm{Sol}_{\Pi_1}$ *such that,* $\forall (x_1, y_2) \in (\mathcal{I}_{\Pi_1} \times \mathrm{Sol}_{\Pi_2}(f(x_1)))$, $g(x_1, y_2) \in \mathrm{Sol}_{\Pi_1}(x_1)$; *in other words, starting from a solution y_2 of the instance x_2, g determines a solution $y_1 = g(x_1, y_2)$ of the initial instance x_1.*

As we informally said in the introduction, the aim of an approximation preserving reduction is to guarantee that if we achieve a certain degree of approximation in the solution of problem Π_2, then a suitable degree of approximation is reached for problem Π_1. As we will see, the various notions of approximation preserving reducibilities that will be introduced in the following, essentially differ in the mapping that is established between the approximation ratios of the two problems.

Before closing this section, let us introduce the notion of *closure* of a class of problems under a given type of reducibility. In what follows, given two **NPO** problems Π and Π', and a reducibility X, we will generally use the notation $\Pi \leq_{\mathsf{X}} \Pi'$ to indicate that Π reduces to Π' via reduction of type X.

Definition 15.7

*Let **C** be a class of **NPO** problems and X a reducibility. Then, the closure* $\overline{\mathbf{C}}^{\mathsf{X}}$ *of **C** under X is defined as:* $\overline{\mathbf{C}}^{\mathsf{X}} = \{\Pi \in \boldsymbol{NPO} : \exists \Pi' \in \mathbf{C}, \Pi \leq_{\mathsf{X}} \Pi'\}$.

15.3 The Linear Reducibility

The first kind of approximation preserving reducibility that we want to show is a very natural and simple transformation among problems which consists of two linear mappings: one between the values of the optimum solutions of the two problems and one between the errors of the corresponding approximate solutions, the *linear reducibility* (L-reducibility, denoted $\leq_{\mathsf{L}}$).

Definition 15.8

Let Π_1 *and* Π_2 *be two problems in **NPO**. Then, we say that* $\Pi_1 \leq_{\mathsf{L}} \Pi_2$*, if there exist two functions* f *and* g *(basic reduction) and two constants* $\alpha_1 > 0$ *and* $\alpha_2 > 0$ *such that* $\forall x \in \mathcal{I}_{\Pi_1}$ *and* $\forall y' \in \mathrm{Sol}_{\Pi_2}(f(x))$*:*

- $\mathrm{opt}_{\Pi_2}(f(x)) \leq \alpha_1 \, \mathrm{opt}_{\Pi_1}(x)$;
- $|m_{\Pi_1}(x, g(y')) - \mathrm{opt}_{\Pi_1}(x)| \leq \alpha_2 |m_{\Pi_2}(f(x), y') - \mathrm{opt}_{\Pi_2}(f(x))|$.

This type of reducibility has been introduced in Ref. [8] and has played an important role in the characterization of the hardness of approximation. In fact it is easy to observe that the following property holds.

Fact 15.1

Given two problems Π *and* Π'*, if* $\Pi \leq_{\mathsf{L}} \Pi'$ *and* $\Pi' \in$ ***PTAS****, then* $\Pi \in$ ***PTAS****. In other words, the* L*-reduction preserves membership in* ***PTAS****.*

Example 15.1

MAX 3-SAT $\leq_{\mathsf{L}}$ MAX 2-SAT. *Let us consider an instance* ϕ *with* m *clauses (w.l.o.g., let us assume that all clauses consist of exactly three literals); let* l_i^1, l_i^2, *and* l_i^3 *be the three literals of the* i*th clause,* $i = 1, \ldots, m$. *To any clause we associate the following 10 new clauses, each one consisting of at most two literals:* $l_i^1, l_i^2, l_i^3, l_i^4, \bar{l}_i^1 \vee \bar{l}_i^2, \bar{l}_i^1 \vee \bar{l}_i^3, \bar{l}_i^2 \vee \bar{l}_i^3, l_i^1 \vee \bar{l}_i^4, l_i^2 \vee \bar{l}_i^4, l_i^3 \vee \bar{l}_i^4$, *where* l_i^4 *is a new variable. Let* C_i' *be the conjunction of the 10 clauses derived from clause* C_i. *The formula* $\phi' = f(\phi)$ *is the conjunction of all clauses* C_i', $i = 1, \ldots, m$, i.e., $\phi' = f(\phi) = \wedge_{i=1}^{m} C_i'$ *and it is an instance of* MAX 2-SAT.

It is easy to see that all truth assignments for ϕ' *satisfy at most seven clauses in any* C_i'. *On the other side, for any truth assignment for* ϕ *satisfying* C_i, *the following truth assignment for* l_i^4 *is such that the extended truth assignment satisfies exactly seven clauses in* C_i': *if exactly one (resp., all) of the variables* l_i^1, l_i^2, l_i^3 *is (resp., are) set to true, then* l_i^4 *is set to false (resp., true); otherwise (exactly one literal in* C_i *is set to false),* l_i^4 *can be indifferently true or false. Finally, if* C_i *is not satisfied (*l_i^1, l_i^2, *and* l_i^3 *are all set to false), no truth assignment for* l_i^4 *can satisfy more than six clauses of* C_i' *while six are guaranteed by setting* l_i^4 *to false. This implies that* $\mathrm{opt}(\phi') = 6m + \mathrm{opt}(\phi) \leq 13\, \mathrm{opt}(\phi)$ *(since* $m \leq 2\, \mathrm{opt}(\phi)$, *see Lemma 15.2 in Section* 15.6.2).

Given a truth assignment for ϕ', *we consider its restriction* $\tau = g(\phi, \tau')$ *on the variables of* ϕ*; for such assignment* τ *we have:* $m(\phi, \tau) \geq m(\phi', \tau') - 6m$. *Then,* $\mathrm{opt}(\phi) - m(\phi, \tau) = \mathrm{opt}(\phi') - 6m - m(\phi, \tau) \leq \mathrm{opt}(\phi') - m(\phi', \tau')$. *This means that the reduction we have defined is an* L*-reduction with* $\alpha_1 = 13$ *and* $\alpha_2 = 1$.

L-reductions provide a simple way to prove hardness of approximability. An immediate consequence of the reduction that has been shown above and of Fact 15.1 is that, since MAX 3-SAT does not allow a PTAS (see Chapter 17) so does MAX 2-SAT. The same technique can be used to show the nonexistence of PTAS for a large class of optimization problems, among others MAX CUT, MAX INDEPENDENT SET-B (i.e., MAX INDEPENDENT SET on graphs with bounded degree), and MIN VERTEX COVER.

Before closing this section, let us observe that the set of 10 2-SAT clauses that we have used in Example 15.1 for constructing the 2-SAT formula $\phi' = f(\phi)$, is strongly related to the bound on approximability established in the example. Really, the proof of the result is based on the fact that at least six out of the 10 clauses can always be satisfied while exactly seven out of 10 can be satisfied, if and only if the original 3-SAT clause is satisfied. A combinatorial structure of this kind, which allows to transfer (in)approximability results from a problem to another, is called a *gadget* (see Ref. [9]). The role of gadgets in approximation-preserving reductions will be discussed further in Section 15.8.

15.4 Strict Reducibility and Complete Problems in NPO

As we informally said in the introduction, an important characteristic of an approximation preserving reduction from a problem Π_1 to a problem Π_2 is that the solution y_1 of problem Π_1 produced by the mapping g should be at least as good as the original solution y_2 of problem Π_2. This property is not necessarily true for any approximation preserving reduction (it is easy to observe that, for example, L-reductions do not always satisfy it), but it is true for the most natural reductions that have been introduced in the early phase of approximation studies: the *strict reductions* [7].

In the following, we present the strict reducibility (S-reducibility, denoted $\leq_S$) referring to minimization problems, but the definition can be trivially extended to all types of optimization problems.

Definition 15.9

*Let Π_1 and Π_2 be two **NPO** minimization problems. Then, we say that $\Pi_1 \leq_S \Pi_2$ if there exist two polynomial-time computable functions f, g that satisfy the following properties:*

- *f and g are defined as in a basic reduction;*
- $\forall x \in \mathcal{I}_{\Pi_1}, \forall y \in \mathrm{Sol}_{\Pi_2}(f(x)), \rho_{\Pi_2}(f(x), y) \geq \rho_{\Pi_1}(x, g(x, y))$.

It is easy to observe that the S-reducibility preserves both membership in **APX** and in **PTAS**.

Proposition 15.1

*Given two minimization problems Π_1 and Π_2, if $\Pi_1 \leq_S \Pi_2$ and $\Pi_2 \in$ **APX** (resp., $\Pi_2 \in$ **PTAS**), then $\Pi_1 \in$ **APX** (resp., $\Pi_2 \in$ **PTAS**).*

Example 15.2

Consider the MIN-WEIGHTED VERTEX COVER *problem in which the weights of vertices are bounded by a polynomial $p(n)$ and let us prove that this problem S reduces to the unweighted* MIN VERTEX COVER *problem. Let us consider an instance $(G(V, E), \vec{w})$ of the former and let us see how it can be transformed into an instance $G'(V', E')$ of the latter. We proceed as follows: for any vertex $v_i \in V$, with weight w_i, we construct an independent set W_i of w_i new vertices in V'; next, for any edge $(v_i, v_j) \in E$, we construct a complete bipartite graph among the vertices of the independent sets W_i and W_j in G'. This transformation is clearly polynomial since the resulting graph G' has $\sum_{i=1}^{n} w_i \leq np(n)$ vertices.*

Let us now consider a cover C' of G' and, w.l.o.g., let us assume it is minimal w.r.t. inclusion (in case it is not, we can easily delete vertices until we reach a minimal cover). We claim that at this point C' has the form: $\cup_{j=1}^{\ell} W_{i_j}$, i.e., there is an ℓ such that C' consists of ℓ independent sets W_i. Suppose that the claim is not true. Let us consider an independent set W_k which is only partially included in C' (that is a nonempty portion W_k' of it belongs to C'). Let us also consider all independent sets W_p that are entirely or partially included in C' and moreover are connected by edges to the vertices of W_k. Two cases may arise: (i) all considered sets W_p have their vertices included in C'; in this case the existence of W_k' would contradict the minimality of C'; (ii) among the considered sets W_p there is at least one set W_q out of which only a nonempty portion W_q' is included in C'; in this case, since the subgraph of G' induced by $W_k \cup W_q$ is a complete bipartite graph, the *edges connecting the vertices of $W_p \setminus W'_p$ with the vertices of $W_q \setminus W'_q$ are not covered by C' and this would contradict the assumption that C' is a cover of G'. As a consequence, the size of C' satisfies $|C'| = \sum_{j=1}^{\ell} w_{i_j}$ and the function g of the reduction can then be defined as follows:* if C' is a cover of G' and if $W_i, i = 1, \ldots, \ell$, are the independent sets that form C', then a cover C for G contains all corresponding vertices $v_1, \ldots, v_\ell$ of V. *Clearly g can be computed in polynomial time.*

From these premises we can immediately infer that the same approximation ratio that is guaranteed for A on G' is also guaranteed by g on G. The shown reduction is hence an S-reduction.

An immediate corollary of the strict reduction shown in the example is that the approximation ratio 2 for MIN VERTEX COVER (that we know can be achieved by various approximation techniques, see Ref. [10]) also holds for the weighted version of the problem, dealt in Example 15.2.

The S-reducibility is indeed a very strong type of reducibility: in fact it requires a strong similarity between two optimization problems and it is not easy to find problems that exhibit such similarity. The interest for the S-reducibility arises mainly from the fact that by making use of reductions of this kind, Orponen and Mannila have identified the first optimization problem that is complete in the class of **NPO** minimization problems: the problem MIN WSAT. Let us consider a Boolean formula in conjunctive normal form ϕ over n variables $x_1, \ldots, x_n$ and m clauses. Any variable x_i has a positive weight $w_i = w(x_i)$. Let us assume that the truth assignment that puts all variables to *true* is feasible, even if it does not satisfy ϕ. Besides, let us assume that t_i is equal to 1 if τ assigns value *true* to the ith variable and 0 otherwise. We want to determine the truth assignment τ of ϕ which minimizes $\sum_{i=1}^{n} w_i t_i$. The problem MAX WSAT can be defined in similar terms. In this case, we assume that the truth assignment that puts all variables to *false* is feasible and we want to determine the truth assignment τ that maximizes $\sum_{i=1}^{n} w_i t_i$. In the variants MIN W3-SAT and MAX W3-SAT, we consider that all clauses contain exactly three literals.

The fact that MIN WSAT is complete in the class of **NPO** minimization problems under S-reductions implies that this problem does not allow any constant-ratio approximation (unless **P** = **NP**) [5–7]. In fact, due to the properties of S-reductions, if a problem which is complete in the class of **NPO** minimization problems was approximable then all **NPO** minimization problems would. Since it is already known that some minimization problems in **NPO** do not allow any constant-ratio approximation algorithm (namely MIN TSP on general graphs), then we can deduce that (unless **P** = **NP**) no complete problem in the class of NPO minimization problems allows any constant-ratio approximation algorithm.

Theorem 15.1

MIN WSAT *is complete in the class of minimization problems belonging to* **NPO** *under S-reductions.*

Proof

The proof is based on a modification of Cook's proof of the **NP**-completeness of SAT [1]. Let us consider a minimization problem $\Pi \in$ **NPO**, the polynomial p which provides the bounds relative to problem Π (see Definition 15.1) and an instance x of Π. The following nondeterministic Turing machine M (with two output tapes T_1 and T_2) generates all feasible solutions $y \in \mathrm{Sol}(x)$ together with their values:

- generate y, such that $|y| \leq p(|x|)$;
- if $y \notin \mathrm{Sol}(x)$, then reject; otherwise, write y on output tape T_1, $m(x, y)$ on output tape T_2, and accept.

Let us now consider the reduction that is currently used in the proof of Cook's theorem (see Ref. [11]) and remember that such reduction produces a propositional formula in conjunctive normal form that is satisfied if and only if the computation of the Turing machine accepts. Let ϕ_x be such formula and $x_n, x_{n-1}, \ldots, x_0$ the variables of ϕ_x that correspond to the cells of tape T_2 where M writes the value $m(x, y)$ in binary (w.l.o.g., we can assume such cells to be consecutive), such that a satisfying assignment of ϕ_x, x_i is *true* if and only if the $(n - i)$-th bit of $m(x, y)$ is equal to 1. Given an instance x of Π the function f of the S-reduction provides an instance of MIN WSAT consisting of the pair (ϕ_x, ψ), where $\psi(x) = \psi\ (x_i) = 2^i$, for $i = 0, \ldots, n$ and $\psi(x) = 0$, for any other variable x in ϕ_x.

The function g of the S-reduction is defined as follows. For any instance x of Π and any solution $\tau' \in \mathrm{Sol}(f(x))$ (i.e., any truth assignment τ' which satisfies the formula ϕ_x [for simplicity we only consider the case in which the formula ϕ_x is satisfiable]), we recover from ϕ_x the representation of the solution y written on tape T_1. Besides, we have that $m(x, g(x, \tau')) = \sum_{\tau'(x_i)=\mathbf{true}} 2^i = m((\phi_x, \psi), \tau')$, where by $\tau'(x_i)$ we indicate the value of variable x_i according to the assignment τ'. As a consequence, $m(x, g(x, \tau')) = m(f(x), \tau')$ and henceforth, $r(x, g(x, \tau')) = r(f(x), \tau')$, and the described reduction is an S-reduction. □

After having established that MIN WSAT is complete for **NPO** minimization problems under the S-reducibility we can then proceed to find other complete problems in this class.

Let us consider the following definition of the MIN 0-1 LINEAR PROGRAMMING problem (the problem MAX 0-1 LINEAR PROGRAMMING can be defined analogously). We consider a matrix $A \in \mathbb{Z}^{m\times n}$ and two vectors $\vec{b} \in \mathbb{Z}^m$ and $\vec{w} \in \mathbb{N}^n$. We want to determine a vector $\vec{y} \in \{0, 1\}^n$ that verifies $A\vec{y} \geq \vec{b}$ and minimizes the quantity $\vec{w} \cdot \vec{y}$.

Clearly, MIN 0-1 LINEAR PROGRAMMING is an **NPO** minimization problem. The reduction from MIN WSAT to MIN 0-1 LINEAR PROGRAMMING is a simple modification of the standard reduction among the corresponding decision problems. Suppose that the following instance of MIN 0-1 LINEAR PROGRAMMING, consisting of a matrix $A \in \mathbb{Z}^{m\times n}$ and two vectors $\vec{b} \in \mathbb{Z}^m$ and $\vec{w} \in \mathbb{N}^n$, is the image $f(x)$ of an instance x of MIN WSAT and suppose that $\vec{y}$ is a feasible solution of $f(x)$ whose value is $m(f(x), \vec{y}) = \vec{w} \cdot \vec{y}$. Then, $g(x, \vec{y})$ is a feasible solution of x, that is a truth assignment τ, whose value is $m(x, \tau) = \sum_{i=1}^{n} w_i t_i$ where t_i is equal to 1 if τ assigns value *true* to the ith variable and 0 otherwise. Since we have $\sum_{i=1}^{n} w_i t_i = \vec{w} \cdot \vec{y}$, it is easy to see that the reduction (f, g, c), where c is the identity function, is an S-reduction[1] and, as a consequence, MIN 0-1 LINEAR PROGRAMMING is also complete in the class of **NPO** minimization problems.

It is not difficult to prove that an analogous result holds for maximization problems, that is, MAX WSAT is complete under S-reductions in the class of **NPO** maximization problems.

At this point of the chapter we still do not have the technical instruments to establish a more powerful result, that is, to identify problems which are complete under S-reductions for the entire class of **NPO** problems. To prove such a result we need to introduce a more involved kind of reducibility, the AP-reducibility (see Section 15.5). In fact, by means of AP-reductions MAX WSAT can itself be reduced to MIN WSAT and vice versa (see Ref. [12]) and therefore it can be shown that (under AP-reductions) both problems are indeed **NPO**-complete.

15.5 AP-Reducibility

After the seminal paper by Orponen and Mannila [7], research on approximation preserving reducibility was further developed (see, e.g., Refs. [13–15]); nevertheless, the beginning of the structural theory of approximability of optimization problems can be traced back to the fundamental paper by Crescenzi and Panconesi [16] where reducibilities preserving membership in **APX** (A-reducibility), **PTAS** (P-reducibility), and **FPTAS** (F-reducibility) were studied and complete problems for each of the three kinds of reducibilities were shown, respectively in **NPO**, **APX**, and **PTAS**. Unfortunately, the problems which are proved complete in **APX** and **PTAS** in this paper are quite artificial.

Along a different line of research, during the same years, the study of logical properties of optimization problems has led Papadimitriou and Yannakakis [8] to the syntactic characterization of an important class of approximable problems, the class **Max-SNP**. Completeness in **Max-SNP** has been defined in terms of L-reductions (see Section 15.3) and natural complete problems (e.g., MAX 3-SAT, MAX 2-SAT, and MIN VERTEX COVER-B) have been found. The relevance of such an approach is related to the fact that it is possible to prove that **Max-SNP**-complete problems do not allow PTAS (unless $\mathbf{P} = \mathbf{NP}$).

[1] Note that, in this case, the reduction is also a linear reductions with $\alpha_1 = \alpha_2 = 1$.

The two approaches have been reconciled by Khanna et al. [17], where the closure of syntactically defined classes with respect to an approximation preserving reduction were proved equal to the more familiar computationally defined classes. As a consequence of this result, any Max-SNP-completeness result appeared in the literature can be interpreted as an APX-completeness result. In this paper a new type of reducibility is introduced, the E-reducibility. With respect to the L-reducibility, in the E-reducibility the constant α_1 is replaced by a polynomial $p(|x|)$. This reducibility is fairly powerful since it allows to prove that MAX 3-SAT is complete for **APX-PB** (the class of problems in **APX** whose values are bounded by a polynomial in the size of the instance) such as MAX 3-SAT. However, it remains somewhat restricted because it does not allow the transformation of **PTAS** problems (such as MAX KNAPSACK) into problems belonging to **APX-PB**.

The final answer to the problem of finding the suitable kind of reducibility (powerful enough to establish completeness results both in **NPO** and **APX**) is the AP-reducibility introduced by Crescenzi et al. [12].

In fact, the types of reducibility that we have introduced so far (linear and strict reducibilities) suffer from various limitations. In particular, we have seen that strict reductions allow us to prove the completeness of MIN WSAT in the class of **NPO** minimization problems, but are not powerful enough to allow the identification of problems which are complete for the entire class **NPO**. Besides, both linear and strict reductions, in different ways, impose strong constraints on the values of the solutions of the problems among which the reduction is established.

In this section, we provide the definition of the AP-reducibility (denoted $\leq_{\mathsf{AP}}$) and we illustrate its properties. Completeness results in **NPO** and in **APX** based on AP-reductions are shown in Section 15.6.

Definition 15.10

*Let Π_1 and Π_2 be two minimization **NPO** problems. An AP-reduction between Π_1 and Π_2 is a triple (f, g, α), where f and g are functions and α is a constant, such that, for any $x \in \mathcal{I}_{\Pi_1}$ and $r > 1$:*

- *$f(x, r) \in \mathcal{I}_{\Pi_2}$ is computable in time $t_f(|x|, r)$ polynomial in $|x|$ for a fixed r; $t_f(n, \cdot)$ is nonincreasing;*
- *for any $y \in \mathrm{Sol}_{\Pi_2}(f(x, r))$, $g(x, y, r) \in \mathrm{Sol}_{\Pi_1}(x)$ is computable in time $t_g(|x|, y, r)$ which is polynomial both in $|x|$ and in $|y|$ for an fixed r; $t_g(n, n, \cdot)$ is nonincreasing;*
- *for any $y \in \mathrm{Sol}_{\Pi_2}(f(x, r))$, $\rho_{\Pi_2}(f(x, r), y) \leq r$ implies $\rho_{\Pi_1}(x, g(x, y, r)) \leq 1 + \alpha(r - 1)$.*

It is worth underlining the main differences of AP-reductions with respect to the reductions introduced until now. First, with respect to L-reductions the constraint that the optimum values of the two problems are linearly related has been dropped. Second, with respect to the S-reductions we allow a weaker relationship to hold between the approximation ratios achieved for the two problems. Besides, an important condition which is needed in the proof of APX-completeness is that, in AP-reductions, the two functions f and g may depend on the approximation ratio r. Such extension is somewhat natural since there is no reason to ignore the quality of the solution we are looking for, when reducing one optimization problem to another and it plays a crucial role in the completeness proofs. However, since in many applications such knowledge is not required, whenever functions f and g do not use the dependency on r, we will avoid specifying this dependency. In other words, we will write $f(x)$ and $g(x, y)$ instead of $f(x, r)$ and $g(x, y, r)$, respectively.

Proposition 15.2

*Given two minimization problems Π_1 and Π_2, if $\Pi_1 \leq_{\mathsf{AP}} \Pi_2$ and $\Pi_2 \in$ **APX** (resp., $\Pi_2 \in$ **PTAS**), then $\Pi_1 \in$ **APX** (resp., $\Pi_1 \in$ **PTAS**).*

As a last remark, let us observe that the S-reducibility is a particular case of AP-reducibility, corresponding to the case in which $\alpha = 1$. More generally, the AP-reducibility is sufficiently broad to encompass almost all known approximation preserving reducibilities while maintaining the property of establishing a linear relation between performance ratios: This is important to preserve membership in all approximation classes.

15.6 NPO-Completeness and APX-Completeness

15.6.1 NPO-Completeness

In the preceding section, we have announced that by means of a suitable type of reduction we can transform an instance of MAX WSAT into an instance of MIN WSAT. This can now be obtained by making use of AP-reductions. By combining this result with Theorem 15.1 and with the corresponding result concerning the completeness of MAX WSAT in the class of **NPO** maximization problems, we can assemble the complete proof that MIN WSAT is complete for the entire class **NPO** under AP-reductions. The inverse reduction, from MIN WSAT to MAX WSAT can be shown in a similar way, leading to the proof that also MAX WSAT is complete for the entire class **NPO** under AP-reductions.

Theorem 15.2

MAX WSAT *can be AP-reduced to* MIN WSAT *and vice versa.*

Proof (Sketch)

The proof works as follows. First a simple reduction can be defined which transforms a given instance ϕ of MAX WSAT into an instance ϕ' of MIN WSAT with α depending on r. Such reduction can then be modified into a real AP-reduction in which α is a constant, not depending on r, while, of course, the functions f and g will depend on r. We limit ourselves to describing the first step. The complete proof can be found in Ref. [18].

Let ϕ be the formula produced in the reduction proving the completeness of MAX WSAT for the class of NPO maximization problems. Then, $f(\phi)$ be the formula $\phi \wedge \alpha_1 \wedge \cdots \wedge \alpha_s$, where α_i is $z_i \equiv (\overline{v}_1 \wedge \cdots \wedge \overline{v}_{i-1} \wedge v_i)$, $z_1, \ldots, z_s$ are new variables with weights $w(z_i) = 2^i$ for $i = 1, \ldots, s$, and all other variables (even the v variables) have zero weight. If τ is a satisfying truth assignment for $f(\phi)$, let $g(\phi, \tau)$ be the restriction of τ to the variables that occur in ϕ. This assignment clearly satisfies ϕ. Note that exactly one among the z variables is *true* in any satisfying truth assignment of $f(\phi)$. If all z variables were *false*, then all v variables would be *false*, which is not allowed. However, it is clearly not possible that two z variables are *true*. Hence, for any feasible solution τ of $f(\phi)$, we have that $m(f(\phi), \tau) = 2^i$, for some i with $1 \leq i \leq s$. This finally implies that $2^s/m(f(\phi), \tau) \leq m(\phi, g(\phi, \tau)) < 2.2^s/m(f(\phi), \tau)$. This is particularly true for the optimal solution (observe that any satisfying truth assignment for ϕ can be easily extended to a satisfying truth assignment for $f(\tau)$). Thus, after some easy algebra, the performance ratio of $g(\phi, \tau)$ with respect to ϕ verifies $r(\phi, g(\phi, \tau)) > 1/(2r(f(\phi), \tau))$.

The reduction satisfies the approximation preserving condition with a factor $\alpha = (2r - 1)/(r - 1)$. To obtain a factor α not depending on r, the reduction can be modified by introducing 2^k more variables for a suitable integer k. □

Other problems that have been shown **NPO**-complete are MIN (MAX) W3-SAT and MIN TSP [7]. As it has been observed before, as a consequence of their **NPO**-completeness under approximation preserving reductions, for all these problems any r-approximate algorithm with constant r does not exist unless **P** = **NP**.

15.6.2 APX-Completeness

As it has been mentioned above, the existence of an **APX**-complete problem has already been shown in Ref. [16] (see also Ref. [19]), but the problem that is proved complete in such framework is a rather artificial version of MAX WSAT. The reduction used in such a result is called P-reduction. Unfortunately, no natural problem has been proved complete in **APX** using the same approach. In this section, we prove the **APX**-completeness under AP-reduction of a natural and popular problem: MAX 3-SAT. The proof is crucially based on the following two lemmas (whose proofs are not provided in this paper).

The first lemma is proved in Ref. [20] and is based on a powerful algebraic technique for the representation of propositional formulæ (see also Ref. [18]), while the second one states a well-known property of propositional formulæ and is proved in Refs. [3,18].

Lemma 15.1

There is a constant $\epsilon > 0$ and two functions f_s and g_s such that, given any propositional formula ϕ in conjunctive normal form, the formula $\psi = f_s(\phi)$ is a conjunctive normal form formula with at most three literals per clause which satisfies the following property: for any truth assignment T' satisfying at least a portion $1 - \epsilon$ of the maximum number of satisfiable clauses in ψ, $g_s(\phi, T')$ satisfies ϕ if and only if ϕ is satisfiable.

Lemma 15.2

Given a propositional formula in conjunctive normal form, at least one-half of its clauses can always be satisfied.

Theorem 15.3

MAX 3-SAT *is **APX**-complete.*

Proof (Sketch)

As it has been done in the case of the proofs of **NPO**-completeness, we split the proof in two parts. First, we show that MAX 3-SAT is complete in the class of **APX** maximization problems and then we show that any **APX** minimization problem can be reduced to an **APX** maximization problem. To make the proof easier, we adopt the convention used in Ref. [18]. The approximation ratio of a maximization problem in this context will be defined as the ratio between the value of the optimum solution opt(x) and the value of the approximate solution $m(x, \mathtt{A}(x))$. For both maximization and minimization problems, therefore, the approximation ratio is in $[1, \infty)$. Let us first observe that MAX 3-SAT $\in$ **APX** since it can be approximated up to the ratio 0.8006 [9].

Now we can sketch the proof that MAX 3-SAT is hard for the class of maximization problems in **APX**. Let us consider a maximization problem $\Pi \in$ **APX**. Let $\mathtt{A}_\Pi$ be a polynomial-time r_Π-approximation algorithm for Π. To construct an AP-reduction, let us define the parameter α as follows: $\alpha = 2(r_\Pi \log r_\Pi + r_\Pi - 1) \times((1 + \epsilon)/\epsilon)$, where ϵ is the constant of Lemma 15.1. Let us now choose $r > 1$ and let us consider the following two cases: $1 + \alpha(r - 1) \geq r_\Pi$ and $1 + \alpha(r - 1) < r_\Pi$.

In the case $1 + \alpha(r - 1) \geq r_\Pi$, given any instance x of Π and given any truth assignment τ for MAX 3-SAT, we trivially define $f(x, r)$ to be the empty formula and $g(x, \tau, r) = \mathtt{A}_\Pi(x)$. It can easily be seen that $r(x, g(x, \tau, r)) \leq r_\Pi \leq 1 + \alpha(r - 1)$ and the reduction is an AP-reduction.

Let us then consider the case $1 + \alpha(r - 1) < r_\Pi$ and let us define $r_n = 1 + \alpha(r - 1)$; then, $r = ((r_n - 1)/\alpha) + 1$. If we define $k = \lceil \log_{r_n} r_\Pi \rceil$, we can partition the interval $[m(x, \mathtt{A}_\Pi(x)), r_\Pi m(x, \mathtt{A}_\Pi(x))]$ in the following k subintervals: $[m(x, \mathtt{A}_\Pi(x)), r_n m(x, \mathtt{A}_\Pi(x))]$, $[r_n^i m(x, \mathtt{A}_\Pi(x)), r_n^{i+1} m(x, \mathtt{A}_\Pi(x))]$, $i = 1, \ldots, k-2$, $[r_n^{k-1} m(x, \mathtt{A}_\Pi(x)), r_\Pi m(x, \mathtt{A}_\Pi(x))]$. Then we have $m(x, \mathtt{A}_\Pi(x)) \leq \text{opt}(x) \leq r_\Pi m(x, \mathtt{A}_\Pi(x)) \leq r_n^k m(x, \mathtt{A}_\Pi(x))$, i.e., the optimum value of instance x of Π belongs to one of the subintervals.

Note that by definition $k < (r_\Pi \log r_\Pi + r_\Pi - 1)/(r_n - 1)$ and by making use of the definitions of α, r, and k, we obtain $r < (\epsilon/(2k(1 + \epsilon))) + 1$.

For any $i = 0, 1, \ldots, k - 1$, let us consider an instance x of Π and the following nondeterministic algorithm, where p is the polynomial that bounds the value of all feasible solutions of Π:

- guess a candidate solution y with value at most $p(|x|)$;
- if $y \in \text{Sol}_\Pi(x)$ and $m_\Pi(x, y) \leq r_n^{i+1} m(x, \mathtt{A}_\Pi(x))$, then return *yes*, otherwise return *no*.

Applying once again the technique of Theorem 15.1, we can construct k propositional formulæ $\phi_0, \phi_1, \ldots, \phi_{k-1}$ such that for any truth assignment τ_i satisfying ϕ_i, $i = 0, 1, \ldots, k - 1$, in polynomial time we can build a feasible solution y of the instance x with $m_\Pi(x, y) \geq r_n^i m(x, \mathtt{A}_\Pi(x))$.

Hence, the instance ψ of MAX 3-SAT that we consider is the following: $\psi = f(x, r) = \wedge_{i=0}^{k-1} f_s(\phi_i)$, where f_s is the function defined in Lemma 15.1; w.l.o.g., we can suppose that all formulæ $f_s(\phi_i)$, $i = 0, \ldots, k - 1$, contain the same number of clauses.

Denote by T a satisfying truth assignment of ψ achieving approximation ratio r and by r_i the approximation ratio guaranteed by τ over $f_s(\phi_i)$. By Lemma 15.2 we get $m(r_i - 1)/(2r_i) \leq \text{opt}(\psi) - m(\psi, T) \leq km(r - 1)/r$. Using this expression for $i = 0, \ldots, k - 1$, we have $m(r_i - 1)/2r_i \leq km(r - 1)/r$, which implies $1 - (2k(r - 1)/r) \leq 1/r_i$ and, finally, $r_i \leq 1 + \epsilon$.

Using Lemma 15.1 again, we derive that, for $i = 0, \ldots, k-1$, the truth assignment $\tau_i = g_s(\phi_i, \tau)$ (where g_s is as defined in Lemma 15.1) satisfies ϕ_i if and only if ϕ_i is satisfiable. Let us call i^* the largest i for which τ_i satisfies ϕ_i; then, $r_n^{i^*} m(x, \mathtt{A}_\Pi(x)) \leq \mathrm{opt}_\Pi(x) \leq r_n^{i^*+1} m(x, \mathtt{A}_\Pi(x))$. Starting from τ_{i^*}, we can then construct a solution y for Π whose value is at least $r_n^{i^*} m(x, \mathtt{A}_\Pi(x))$. This means that y guarantees an approximation ratio r_n. In other words, $r(x, y) \leq r_n = 1 + \alpha(r-1)$ and the reduction (f, g, α) that we have just defined (where g consists in applying g_s, determining i^* and constructing y starting from τ_{i^*}) is an AP-reduction.

Since Π is any maximization problem in **APX**, the completeness of MAX 3-SAT for the class of maximization problems in **APX** follows.

We now turn to the second part of the theorem. In fact, we still have to prove that all minimization problems in **APX** can be AP-reduced to maximization problems and, henceforth, to MAX 3-SAT.

Let us consider a minimization problem $\Pi \in$ **APX** and an algorithm $\mathtt{A}$ with approximation ratio r for Π; let $k = \lceil r \rceil$. We can construct a maximization problem $\Pi' \in$ **APX** and prove that $\Pi \leq_{\mathsf{AP}} \Pi'$. The two problems have the same instances and the same feasible solutions, while the objective function of Π' is defined as follows: given an instance x and a feasible solution y of x, $m_{\Pi'}(x, y) = (k+1)m_\Pi(x, \mathtt{A}(x)) - km_\Pi(x, y)$, if $m_\Pi(x, y) \leq m_\Pi(x, \mathtt{A}(x))$, $m_{\Pi'}(x, y) = m_\Pi(x, \mathtt{A}(x))$, otherwise.

Clearly, $m_\Pi(x, \mathtt{A}(x)) \leq \mathrm{opt}_{\Pi'}(x) \leq (k+1)m_\Pi(x, \mathtt{A}(x))$ and, by definition of Π', the algorithm $\mathtt{A}$ is also an approximation algorithm for this problem with approximation ratio $k+1$; therefore, $\Pi' \in$ **APX**. The reduction from Π to Π' can now be defined as follows: for any instance x of Π, $f(x) = x$; for any instance x of Π and for any solution y of instance $f(x)$ of Π', $g(x, y) = y$, if $m_\Pi(x, y) \leq m_\Pi(x, \mathtt{A}(x))$, $g(x, y) = \mathtt{A}(x)$, otherwise $\alpha = k+1$. Note that f and g do not depend on the approximation ratio r.

We now show that the reduction we have just defined is an AP-reduction. Let y be an r'-approximate solution of $f(x)$; we have to show that the ratio $r_\Pi(x, g(x, y))$ of the solution $g(x, y)$ of the instance x of Π is smaller than, or equal to, $1 + \alpha(r'-1)$. We have the following two cases: $m_\Pi(x, y) \leq m_\Pi(x, \mathtt{A}(x))$ and $m_\Pi(x, y) > m_\Pi(x, \mathtt{A}(x))$.

In the case $m_\Pi(x, y) \leq m_\Pi(x, \mathtt{A}(x))$, we can derive $m_\Pi(x, y) \leq (1 + \alpha(r'-1))\,\mathrm{opt}_\Pi(x)$. In other words, $r_\Pi(x, g(x, y)) = r_\Pi(x, y) \leq 1 + \alpha(r'-1)$.

In the case $m_\Pi(x, y) > m_\Pi(x, \mathtt{A}(x))$, since $\alpha \geq 1$, we have $r_\Pi(x, g(x, y)) = r_\Pi(x, \mathtt{A}(x)) = r_{\Pi'}(x, y) \leq r' \leq 1 + \alpha(r'-1)$.

In conclusion, all minimization problems in **APX** can be AP-reduced to maximization problems in APX and all maximization problems in **APX** can be AP-reduced to MAX 3-SAT. Since the AP-reduction is transitive, the **APX**-completeness of MAX 3-SAT is proved. □

15.6.3 Negative Results Based on APX-Completeness

Similar to what we saw for completeness in **NPO**, also completeness in **APX** implies negative results in terms of approximability of optimization problems. In fact if we could prove that an **APX**-complete problem admits a PTAS, then so would all problems in **APX**. However, it is well known that, unless **P** = **NP**, there are problems in **APX** that do not admit a PTAS (one example for all, MIN SCHEDULING ON IDENTICAL MACHINES, see Ref. [18]), therefore, under the same complexity theoretic hypothesis, no **APX**-complete problem admits a PTAS.

As a consequence of the results in the previous subsection, we can therefore assert that, unless **P** = **NP**, MAX 3-SAT does not admit a PTAS, neither do all other optimization problems that have been shown **APX**-complete (MAX 2-SAT, MIN VERTEX COVER, MAX CUT, MIN METRIC TSP, etc.).

Note that the inapproximability of MAX 3-SAT has been proved by Arora et al. [20] in a breakthrough paper by means of sophisticated techniques based on the concept of *probabilistically checkable proofs*, without any reference to the notion of **APX**-completeness. This fact, though, does not diminish the relevance of approximation preserving reductions and the related completeness notion. In fact, most results that state the nonexistence of PTAS for **APX** optimization problems have been proved starting from MAX 3-SAT, via approximation preserving reductions that allow to carry over the inapproximability results from one problem to another. Second, it is worth noting that the structure of approximation classes with respect to approximation preserving reductions is richer than it appears from this chapter. For example, beside complete problems, other classes of problems can be defined inside approximation classes, identifying the so called *intermediate* problems (see Ref. [18]).

15.7 FT-Reducibility

As we have already pointed out in Section 15.5, **PTAS**-completeness has been studied in Ref. [16] under the so-called F-reduction, preserving membership in **FPTAS**. Under this type of reducibility, a single problem, a rather artificial version of MAX WSAT has been shown **PTAS**-complete. In fact, F-reducibility is quite restrictive since it mainly preserves optimality, henceforth, existence of a **PTAS**-complete polynomially bounded problem is very unlikely.

In Ref. [21], a more "flexible" type of reducibility, called FT-reducibility has been introduced. It is formally defined as follows.

Definition 15.11

Let Π and Π' be two maximization integer-valued problems. Then, Π FT-reduces to Π' (denoted by $\Pi \leq_{\mathrm{FT}} \Pi'$) if, for any $\epsilon > 0$, there exist an oracle $\bigcirc_\alpha^{\Pi'}$ for Π' and an algorithm $\mathtt{A}_\epsilon$ calling $\bigcirc_\alpha^{\Pi'}$ such that

- *$\bigcirc_\alpha^{\Pi'}$ produces, for any $\alpha \in [0, 1]$ and for any instance x' of Π', a feasible solution $\bigcirc_\alpha^{\Pi'}(x')$ of x' that is an $(1 - \alpha)$-approximation;*
- *for any instance x of Π, $y = \mathtt{A}_\epsilon(\bigcirc_\alpha^{\Pi'}, x) \in \mathrm{Sol(x)}$; furthermore the approximation ratio of y is at least $(1 - \epsilon)$;*
- *if $\bigcirc_\alpha^{\Pi'}(\cdot)$ runs in time polynomial in both $|f(x)|$ and $1/\alpha$, then $\mathtt{A}_\epsilon(\bigcirc_\alpha^{\Pi'}(f(x)), x)$ is polynomial in both $|x|$ and $1/\epsilon$.*

For the case where at least one among Π and Π' is a minimization problem it suffices to replace $1 - \epsilon$ or/and $1 - \alpha$ by $1 + \epsilon$ or/and $1 + \alpha$, respectively.

As one can see from Definition 15.11, FT-reduction is somewhat different from the other ones considered in this chapter and, in any case, it is not conformal to Definition 15.6. In fact, it resembles a Turing-reduction. Clearly, FT-reduction transforms an FPTAS for Π' into an FPTAS for Π, i.e., it preserves membership in **FPTAS**. Note also that the F-reduction, as it is defined in Ref. [16], is a special case of the FT-reduction, since the latter explicitly allows multiple calls to oracle $\bigcirc$ while for the former this fact is not explicit.

Theorem 15.4

*Let Π' be an **NP**-hard problem in **NPO**. If $\Pi' \in$ **NPO-PB** (the class of problems in **NPO** whose values are bounded by a polynomial in the size of the instance), then any **NPO** problem FT reduces to Π'. Consequently, (i) $\overline{\mathbf{PTAS}}^{\mathrm{FT}} = \mathbf{NPO}$ and (ii) any **NP**-hard polynomially bounded problem in **PTAS** is **PTAS**-complete under FT-reductions.*

Proof (Sketch)

We first prove the following claim: *if an **NPO** problem Π' is **NP**-hard, then any **NPO** problem Turing reduces (see Ref. [18]) to Π'.*

To prove this claim, let Π be an **NPO** problem and q be a polynomial such that $|y| \leqslant q(|x|)$, for any instance x of Π and for any feasible solution y of x. Assume that the encoding $n(y)$ of y is binary. Then $0 \leq n(y) \leq 2^{q(|x|)} - 1$. We consider problem $\hat{\Pi}$ which is the same as Π up to its value that is defined by $m_{\hat{\Pi}}(x, y) = 2^{q(|x|)+1} m_{\Pi}(x, y) + n(y)$. If $m_{\hat{\Pi}}(x, y_1) \geq m_{\hat{\Pi}}(x, y_2)$, then $m_{\Pi}(x, y_1) \geq m_{\Pi}(x, y_2)$. So, if a solution y is optimal for x, with respect to $\hat{\Pi}$, it is so with respect to Π. Remark now that $\hat{\Pi}$ and its evaluation version $\hat{\Pi}_e$ are equivalent since given the value of an optimal solution y, one can determine $n(y)$ (hence y) by computing the remainder of the division of this value by $2^{q(|x|)+1}$. Since Π' is **NP**-hard, it can be shown that one can solve the evaluation problem $\hat{\Pi}_e$, henceforth $\hat{\Pi}$ if one can solve, the (constructive) problem Π' and the claim is proved.

We now prove the following claim: *let* $\Pi' \in$ ***NPO-PB****; then, any* ***NPO*** *problem Turing-reducible to* Π' *is also FT-reducible to* Π'.

To prove this second claim, let Π be an **NPO** problem and suppose that there exists a Turing-reduction between Π and Π'. Let $\bigcirc_{\alpha}^{\Pi'}$ be as in Definition 15.11. Moreover, let p be a polynomial such that for any instance x' of Π' and for any feasible solution y' of x', $m(x', y') \leqslant p(|x'|)$. Let x be an instance of Π. The Turing-reduction claimed gives an algorithm solving Π using an oracle for Π'. Consider now this algorithm where we use, for any query to the oracle with the instance x' of Π', the approximate oracle $\bigcirc_{\alpha}^{\Pi'}(x')$, with $\alpha = 1/(p(|x'|) + 1)$. This algorithm is polynomial and produces an optimal solution, since a solution y' being an $(1 - (1/(p(|x'|) + 1)))$-approximation for x' is an optimal one. So, the claim is proved.

From the combination of the above claims the theorem is easily derived. □

Observe finally that MAX PLANAR INDEPENDENT SET and MIN PLANAR VERTEX COVER are in both PTAS [22] and **NPO-PB**. So, the following theorem concludes this section.

Theorem 15.5

MAX PLANAR INDEPENDENT SET *and* MIN PLANAR VERTEX COVER *are* ***PTAS****-complete under FT-reductions.*

15.8 Gadgets, Reductions, and Inapproximability Results

As it has been pointed out already in Section 15.3, in the context of approximation preserving reductions, we call *gadget* a combinatorial structure which allows to transfer approximability (or inapproximability) results from a problem to another. A classical example is the set of 10 2-SAT clauses that we have used in Example 15.1 for constructing the 2-SAT formula starting from a 3-SAT formula. Although gadgets are used since the seminal work of Karp on reductions among combinatorial problems, the study of gadgets has been started in Refs. [9,23]; from the latter derive most of the results discussed in this section.

To understand the role of gadgets in approximation preserving reductions, let us first go back to linear reductions and see what are the implications on the approximation ratio of two problems Π and Π', deriving from the fact that $\Pi \leq_{\mathrm{L}} \Pi'$. Suppose Π and Π' are minimization problems, f, g, α_1, and α_2 are the functions and constants that define the linear reduction, x is an instance of problem Π, $f(x)$ is the instance of problem Π' determined by the reduction, and y is a solution of $f(x)$. Then, the following relationship holds between the approximation ratios of Π and Π': $r_{\Pi}(x, g(x, y)) \leq 1 + \alpha_1\alpha_2(r_{\Pi'}(f(x), y) - 1)$, and, therefore, we have that $r_{\Pi'} \leq 1 + (r - 1)/(\alpha_1\alpha_2)$ implies $r_{\Pi} \leq r$.

In the particular case of the reduction between MAX 3-SAT and MAX 2-SAT, we have $\alpha_1\alpha_2 = 13$ and, therefore, we can infer the following results on the approximability upper bounds and lower bounds of the two problems, which may be proved by a simple calculation:

- Since it is known that MAX 2-SAT can be approximated with the ratio 0.931 [24], then MAX 3-SAT can be approximated with ratio 0.103.
- Since it is known that MAX 3-SAT cannot be approximated beyond the threshold 7/8, then MAX 2-SAT cannot be approximated beyond the threshold 103/104.

Although better bounds are now known for these problems (see Karloff and Zwick [25]), it is important to observe that the above given bounds may be straightforwardly derived from the linear reduction between the two problems and are useful to show the role of gadgets. In such reduction, the structure of the gadget is crucial (it determines the value α_1) and it is clear that better bounds could be achieved if the reduction could make use of "smaller" gadgets. In fact, in Ref. [9], by cleverly constructing a more sophisticated type of gadget (in which, in particular, clauses have real weights), the authors derive a 0.801 approximation algorithm for MAX 3-SAT, improving on previously known bounds.

Based on Ref. [23], in Ref. [9] the notion of α gadget (i.e., gadget with performance α) is abstracted and formalized with reference to reductions among constraint satisfaction problems. In the same paper, it is shown that, under suitable circumstances, the search for (possibly optimum) gadgets to be used in approximation preserving reductions, can be pursued in a systematic way by means of a computer program. An example of the results that may be achieved in this way is the following.

Let PC_0 and PC_1 be the families of constraints over three binary variables defined as $\text{PC}_i(a, b, c) = 1$, if $a \oplus b \oplus c = i$, $\text{PC}_i(a, b, c) = 0$, otherwise, and let DICUT be the family of constraints corresponding to directed cuts in a graph. There exists optimum 6.5 gadgets (automatically derived by the computer program) reducing PC_0 and PC_1 to DICUT. As a consequence, for any $\epsilon > 0$, MAX DICUT is hard to approximate to within $12/13 + \epsilon$.

15.9 Conclusion

A large number of other approximation preserving reductions among optimization problems, besides those introduced in this chapter, have been introduced throughout the years. Here we have reported only the major developments. Other overviews of the world of approximation preserving reductions can be found in Refs. [12,26].

As we have already pointed out in Section 15.2, we have not dealt in this chapter with approximability classes beyond **APX**, even if intensive studies have been performed, mainly for **Poly APX**. In Ref. [17], completeness results are established, under the E-reduction, for **Poly-APX-PB** (the class of problems in **Poly APX** whose values are bounded by a polynomial in the size of the instance). Indeed, as we have already discussed in Section 15.5, use of restrictive reductions as the E-reducibility, where the functions f and g do not depend on any parameter ϵ seems very unlikely to be able to handle **Poly-APX**-completeness. As it is shown in Ref. [21] (see also Chapter 16), completeness for the whole **Poly APX** can be handled, for instance, by using PTAS-reduction, a further relaxation of the AP-reduction where the dependence between the approximation ratios of Π and Π' is not restricted to be linear [27]. Under PTAS-reduction, MAX INDEPENDENT SET is **Poly-APX**-complete [21].

Before concluding, it is worth noting that a structural development (based on the definition of approximability classes, approximation preserving reductions, and completeness results), analogous to the one that has been carried on for the classical approach to the theory of approximation, has been elaborated also for the differential approach (see Chapter 16 for a survey). In Refs.[21,28] the approximability classes **DAPX**, **Poly-DAPX** and **DPTAS** are introduced, suitable approximation preserving reductions are defined and complete problems in **NPO**, **DAPX**, **Poly-DAPX**, and **DPTAS**, under such kind of reductions, are shown.

References

[1] Cook, S. A., The complexity of theorem-proving procedures, *Proc. of STOC'71*, 1971, p. 151.

[2] Karp, R. M., Reducibility among combinatorial problems, in *Complexity of Computer Computations*, Miller, R. E. and Thatcher, J. W., Plenum Press, New York, 1972, p. 85.

[3] Johnson, D. S., Approximation algorithms for combinatorial problems, *J. Comput. Syst. Sci.*, 9, 256, 1974.

[4] Ausiello, G., D'Atri, A., and Protasi, M., Structure preserving reductions among convex optimization problems, *J. Comput. Syst. Sci.*, 21, 136, 1980.

[5] Ausiello, G., D'Atri, A., and Protasi, M., Lattice-theoretical ordering properties for NP-complete optimization problems, *Fundamenta Informaticæ*, 4, 83, 1981.

[6] Paz, A. and Moran, S., Non deterministic polynomial optimization problems and their approximations, *Theor. Comput. Sci.*, 15, 251, 1981.

[7] Orponen, P. and Mannila, H., On Approximation Preserving Reductions: Complete Problems and Robust Measures, Technical report C-1987-28, Department of Computer Science, University of Helsinki, Finland, 1987.

[8] Papadimitriou, C. H. and Yannakakis, M., Optimization, approximation and complexity classes, *J. Comput. Syst. Sci.*, 43, 425, 1991.

[9] Trevisan, L., Sorkin, G. B., Sudan, M., and Williamson, D. P., Gadgets, approximation, and linear programming, *SIAM J. Comput.*, 29(6), 2074, 2000.

[10] Garey, M. R. and Johnson, D. S., *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[11] Papadimitriou, C. H., *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[12] Crescenzi, P., Kann, V., Silvestri, R., and Trevisan, L., Structure in approximation classes, *SIAM J. Comput.*, 28(5), 1759, 1999.

[13] Simon, H. U., Continuous reductions among combinatorial optimization problems, *Acta Informatica*, 26, 771, 1989.

[14] Simon, H. U., On approximate solutions for combinatorial optimization problems, *SIAM J. Disc. Math.*, 3(2), 294, 1990.

[15] Krentel, M. W., The complexity of optimization problems, *J. Comput. Syst. Sci.*, 36, 490, 1988.

[16] Crescenzi, P. and Panconesi, A., Completeness in approximation classes, *Inf. Comput.*, 93(2), 241, 1991.

[17] Khanna, S., Motwani, R., Sudan, M., and Vazirani, U., On syntactic versus computational views of approximability, *SIAM J. Comput.*, 28, 164, 1998.

[18] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M., *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties*, Springer, Berlin, 1999.

[19] Ausiello, G., Crescenzi, P., and Protasi, M., Approximate solutions of NP optimization problems, *Theor. Comput. Sci.*, 150, 1, 1995.

[20] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M., Proof verification and intractability of approximation problems, *Proc. of FOCS'92*, 1992, p. 14.

[21] Bazgan, C., Escoffier, B., and Paschos, V. Th., Completeness in standard and differential approximation classes: poly-(D)APX- and (D)PTAS-completeness, *Theor. Comput. Sci.*, 339, 272, 2005.

[22] Baker, B. S., Approximation algorithms for NP-complete problems on planar graphs, *J. Assoc. Comput. Mach.*, 41(1), 153, 1994.

[23] Bellare, M., Goldreich, O., and Sudan, M., Free bits and non-approximability—towards tight results, *SIAM J. Comput.*, 27(3), 804, 1998.

[24] Feige, U. and Goemans, M. X., Approximating the value of two prover proof systems, with applications to max 2sat and max dicut, *Proc. 3rd Israel Symp. on Theory of Computing and Systems, ISTCS'95*, 1995, p. 182.

[25] Karloff, H. and Zwick, U., A 7/8-approximation for MAX 3SAT?, *Proc. of FOCS'97*, 1997, p. 406.

[26] Crescenzi, P., A short guide to approximation preserving reductions, *Proc. Conf. on Computational Complexity*, 1997, p. 262.

[27] Crescenzi, P. and Trevisan, L., On approximation scheme preserving reducibility and its applications, *Theor. Comput. Syst.*, 33(1), 1, 2000.

[28] Ausiello, G., Bazgan, C., Demange, M., and Paschos, V. Th., Completeness in differential approximation classes, *IJFCS*, 16(6), 1267, 2005.

16

Differential Ratio Approximation

Giorgio Ausiello
University of Rome
"La Sapienza"

Vangelis Th. Paschos
LAMSADE CNRS UMR 7024 and
University of Paris–Dauphine

16.1 Introduction

In this chapter we introduce the so-called differential approximation ratio as a measure of the quality of the solutions obtained by approximation algorithms. After providing motivations and basic definitions we show examples of optimization problems for which the evaluation of approximation algorithms based on the differential ratio appears to be more meaningful than the usual approximation ratio used in the classical approach to approximation algorithms. Finally, we discuss some structural results concerning approximation classes based on the differential ratio. Throughout the chapter we make use of the notations introduced in Chapter 15. Also, given an approximation algorithm $\mathtt{A}$ for an **NP** optimization problem Π (the class of these problems is called **NPO**), we denote by $m_{\mathtt{A}}(x, y)$, the value of the solution y computed by $\mathtt{A}$ on instance x of Π. When clear from the context, reference to $\mathtt{A}$ will be omitted. The definitions of most of the problems dealt in this chapter can be found in Refs. [1,2]; also, for graph-theoretic notions, interested readers are referred to Ref. [3].

In several cases, the commonly used approximation measure (called *standard approximation ratio* in what follows) may not be very meaningful in characterizing the quality of approximation algorithms. This happens, in particular, when the ratio of $m(x, y_w)$, the value of the worst solution for a given input x, to the value of the optimum solution $\text{opt}(x)$ is already bounded (above, if $\text{goal}(\Pi) = \min$, below, otherwise). Consider, for instance, the basic maximal matching algorithm for MIN VERTEX COVER that achieves approximation ratio 2. In this algorithm, given a graph $G(V, E)$, a maximal[1] matching M of G is computed and the endpoints of the edges in M are added in the solution for MIN VERTEX COVER. If M is perfect (almost any graph, even relatively sparse, admits a perfect matching [4]), then the whole of V will be included in the cover, while an optimum cover contains at least a half of V. So, in most cases, the absolutely worst solution (that one could compute without using any algorithm) achieves approximation ratio 2.

The remark above is just one of the drawbacks of the standard approximation ratio. Various other drawbacks have been also observed, for instance, the artificial dissymmetry between "equivalent"

[1] With respect to inclusion.

minimization and maximization problems (e.g., MAX CUT and MIN CLUSTERING, see Ref. [5]) introduced by the standard approximation ratio. The most blatant case of such dissymmetry is the one appearing when dealing with the approximation of MIN VERTEX COVER and MAX INDEPENDENT SET (given a graph, a vertex cover is the complement of an independent set with respect to the vertex set of the graph). In other words, using linear programming vocabulary, the objective function of the former is an affine transformation of the objective function of the latter. This equivalence under such simple affine transformation does not reflect in the approximability of these problems in the classical approach: the former problem is approximable within constant ratio, in other words it belongs to the class **APX** of problems that are approximable within constant ratios (see Chapter 15 for definitions of approximability classes based on the standard approximation paradigm; the ones based on the differential paradigm are defined analogously in this chapter, see Section 16.5), while the latter is inapproximable within ratio $\Omega(n^{\epsilon-1})$, for any $\epsilon > 0$ (see Ref. [6] and Chapter 17). In other words, the standard approximation ratio is unstable under affine transformations of the objective function.

To overcome these phoenomena, several researchers have tried to adopt alternative approximation measures not suffering from these inconsistencies. One of them is the ratio $\delta(x, y) = (\omega(x) - m(x, y))/(\omega(x) - \text{opt}(x))$, called differential ratio in the sequel, where $\omega(x)$ is the value of a worst solution for x, called *worst value*. It will be formally dealt in the next sections. It has been used rather punctually and without following a rigorous axiomatic approach until the paper in Ref. [7] where such an approach is formally defined. To our knowledge, differential ratio is introduced in Ref. [8] in 1977, and Refs. [9–11] are, to our knowledge, the most notable cases in which this approach has been applied. It is worth noting that in Ref. [11], a weak axiomatic approach is also presented.

Finally, let us note that several other authors that have also recognized the methodological problems implied by the standard ratio, have proposed other alternative ratios. It is interesting to remark that, in most cases, the new ratios are very close, although with some small or less small differences, to the differential ratio. For instance, in Ref. [12], for studying MAX TSP, it is proposed that the ratio $d(x, y, z_r) = |\,\text{opt}(x) - m(x, y)|/|\,\text{opt}(x) - z_r|$, where z_r is a positive value computable in polynomial time, called *reference value*. It is smaller than the value of any feasible solution of x, hence smaller than $\omega(x)$ (for a maximization problem a worst solution is the one of the smallest feasible value). The quantities $|\,\text{opt}(x) - m(x, y)|$ and $|\,\text{opt}(x) - z_r|$ are called *deviation* and *absolute deviation*, respectively. The approximation ratio $d(x, y, z_r)$ depends on both x and z_r, in other words, there exist a multitude of such ratios for an instance x of an **NPO** problem, one for any possible value of z_r. Consider a maximization problem Π and an instance x of Π. Then, $d(x, y, z_r)$ is increasing with z_r, so, $d(x, y, z_r) \leq d(x, y, \omega(x))$. In fact, in this case, for any reference value z_r: $r(x, y) \geq 1 - d(x, y, z_r) \geq 1 - d(x, y, \omega(x)) = \delta(x, y)$, where r denotes the standard approximation ratio for Π. When $\omega(x)$ is computable in polynomial time, $d(x, y, \omega(x))$ is the smallest (tightest) over all the d-ratios for x. In any case, if for a given problem, one sets $z_r = \omega(x)$, then $d(x, y, \omega(x)) = 1 - \delta(x, y)$ and both ratios have the natural interpretation of estimating the relative position of the approximate solution-value in the interval worst solution-value—optimal value.

16.2 Toward a New Measure of Approximation Paradigm

In Ref. [7], the task of adopting is undertaken, in an axiomatic way, an approximation measure founded on both intuitive and mathematical links between optimization and approximation. It is claimed there that a "consistent" ratio must be *order preserving* (i.e., the better the solution the better the approximation ratio achieved) and *stable under affine transformation of the objective function*. Furthermore, it is proved that no ratio function of two parameters—for example, m, opt—can fit this latter requirement. Hence, it is proposed what will be called *differential approximation ratio*[2] in what follows. Problems related by affine transformations of their objective functions are called *affine equivalent*.

[2] This notation is suggested in Ref. [7]; another notation drawing the same measure is *z-approximation* suggested in Ref. [13].

Consider an instance x of an **NPO** problem Π and a polynomial-time approximation algorithm $\mathtt{A}$ for Π, the differential approximation ratio $\delta_{\mathtt{A}}(x, y)$ of a solution y computed by $\mathtt{A}$ in x is defined by: $\delta_{\mathtt{A}}(x, y) = (\omega(x) - m_{\mathtt{A}}(x, y))/(\omega(x) - \mathrm{opt}(x))$, where $\omega(x)$ is the value of a worst solution for x, called *worst value*. Note that for any goal, $\delta_{\mathtt{A}}(x, y) \in [0, 1]$ and, moreover, the closer $\delta_{\mathtt{A}}(x, y)$ to 1, the closer $m_{\mathtt{A}}(x, y)$ to $\mathrm{opt}(x)$. By definition, when $\omega(x) = \mathrm{opt}(x)$, i.e., all the solutions of x have the same value, then the approximation ratio is 1. Note that, $m_{\mathtt{A}}(x, y) = \delta_{\mathtt{A}}(x, y)\,\mathrm{opt}(x) + (1 - \delta_{\mathtt{A}}(x, y))\omega(x)$. So, differential approximation ratio measures how an approximate solution is placed in the interval between $\omega(x)$ and $\mathrm{opt}(x)$.

We note that the concept of the worst solution has a status similar to the optimum solution. It depends on the problem itself and is defined in a nonconstructive way, i.e., independently of any algorithm that could build it. The following definition for worst solution is proposed in Ref. [7].

Definition 16.1

*Given an **NPO** problem $\Pi = (\mathcal{I}, \mathrm{Sol}, m, \mathrm{goal})$, a worst solution of an instance x of Π is defined as an optimum solution of a new problem $\bar{\Pi} = (\mathcal{I}, \mathrm{Sol}, m, \overline{\mathrm{goal}})$, i.e., of an **NPO** problem having the same sets of instances and of instances and of feasible solutions and the same value-function as Π but its goal is the inverse w.r.t. Π, i.e., $\overline{\mathrm{goal}} = \min$ if $\mathrm{goal} = \max$ and vice versa.*

Example 16.1

The worst solution for an instance of MIN VERTEX COVER or of MIN COLORING is the whole vertex set of the input graph, while for an instance of MAX INDEPENDENT SET the worst solution is the empty set. However, if one deals with MAX INDEPENDENT SET with the additional constraint that a feasible solution has to be maximal with respect to inclusion, the worst solution of an instance of this variant is a *minimum–maximal independent set*, i.e., an optimum solution of a very well-known combinatorial problem, the MIN INDEPENDENT DOMINATING SET. Also, the worst solution for MIN TSP is a "heaviest" Hamiltonian cycle of the input graph, i.e., an optimum solution of MAX TSP, while for MAX TSP the worst solution is the optimum solution of a MIN TSP. The same holds for the pair MAX SAT, MIN SAT.

From Example 16.1, one can see that, although for some problems a worst solution corresponds to some trivial input parameter and can be computed in polynomial time (this is, for instance, the case with MIN VERTEX COVER, MAX INDEPENDENT SET, MIN COLORING, etc.), several problems exist for which determining a worst solution is as hard as determining an optimum one (as for MIN INDEPENDENT DOMINATING SET, MIN TSP, MAX TSP, MIN SAT, MAX SAT, etc.).

Remark 16.1

Consider the pair of affine equivalent problems MIN VERTEX COVER, MAX INDEPENDENT SET, *and an input graph $G(V, E)$ of order n. Denote by $\tau(G)$ the cardinality of a minimum vertex cover of G and by $\alpha(G)$, the stability number of G. Obviously, $\tau(G) = n-\alpha(G)$. Based upon what has been discussed above, the differential ratio of some vertex cover C of G is $\delta(G, C) = (n-|C|)/(n-\tau(G))$. Since the set $S = V \setminus C$ is an independent set of G, its differential ratio is $\delta(G, S) = (|S| - 0)/(\alpha(G) - 0) = (n - |C|)/(n - \tau(G)) = \delta(G, C)$.*

As we have already mentioned, the differential ratio, although not systematically, has been used several times by many authors, before and after [7], in various contexts going from mathematical (linear or nonlinear) programming [14–16] to pure combinatorial optimization [9,10,13,17,18]. Sometimes the use of the differential approach has been disguised by considering the standard approximation ratio of affine transformations of a problem. For instance, to study differential approximation of BIN PACKING, one can deal with standard approximation of the problem of maximizing the number of unused bins; for MIN COLORING, the affinely equivalent problem is the one of maximizing the number of unused colors, for MIN SET COVER, the problem consists in maximizing the number of unused sets, etc.

16.3 Differential Approximation Results for Some Optimization Problems

In general, no systematic way allows to link results obtained in standard and differential approximation paradigms when dealing with minimization problems. In other words, there is no evident transfer of positive or inapproximability results from one framework to the other. Hence, a "good" differential approximation result does not signify anything for the behavior of the approximation algorithm studied, or of the problem itself, when dealing with the standard framework, and vice versa. Things are somewhat different for maximization problems with positive solution-values. In fact, considering an instance x of a maximization problem Π and a solution $y \in \text{Sol}(x)$ that is a δ-differential approximation, we immediately get:

$$\frac{m(x, y) - \omega(x)}{\text{opt}(x) - \omega(x)} \geq \delta \Longrightarrow \frac{m(x, y)}{\text{opt}(x)} \geqslant \delta + (1 - \delta)\frac{\omega(x)}{\text{opt}(x)} \overset{\omega(x)\geqslant 0}{\Longrightarrow} \frac{m(x, y)}{\text{opt}(x)} \geqslant \delta$$

So, positive results are transferred from differential to standard approximation, while transfer of inapproximability thresholds is done in the opposite direction.

Fact 16.1

Approximation of a maximization **NPO** *problem* Π *within differential approximation ratio* δ*, implies its approximation within standard approximation ratio* δ*.*

Fact 16.1 has interesting applications. The most immediate of them deals with the case of maximization problems with worst-solution values 0. There, standard and approximation ratios coincide. In this case, the differential paradigm inherits the inapproximability thresholds of the standard one. For instance, the inapproximability of MAX INDEPENDENT SET within $n^{\epsilon-1}$, for any $\epsilon > 0$ [6], also holds in the differential approach.

Furthermore, since MAX INDEPENDENT SET and MIN VERTEX COVER are affine equivalent, henceforth differentially equiapproximable, the negative result for MAX INDEPENDENT SET is shared, in the differential paradigm, by MIN VERTEX COVER.

Corollary 16.1

Both MAX INDEPENDENT SET *and* MIN VERTEX COVER *are inapproximable within differential ratios* $n^{\epsilon-1}$*, for any* $\epsilon > 0$*, unless* $\mathbf{P} = \mathbf{NP}$*.*

Note that differential equi-approximability of MAX INDEPENDENT SET and MIN VERTEX COVER makes that, in this framework the latter problem is not constant approximable but inherits also the positive standard approximation results of the former one [19–21].

In what follows in this section, we mainly focus ourselves on three well-known **NPO** problems: MIN COLORING, BIN PACKING, TSP in both minimization and maximization variants, and MIN MULTIPROCESSOR SCHEDULING. As we will see, approximabilities of MIN COLORING and MIN TSP are radically different from the standard paradigm (where these problems are very hard) to the differential one (where they become fairly well approximable). For the first two of them, differential approximability will be introduced by means of more general problem that encompasses both MIN COLORING and BIN PACKING, namely, the MIN HEREDITARY COVER.

16.3.1 Min Hereditary Cover

Let π be a nontrivial *hereditary* property[3] on sets and C a ground set. A π-covering of C is a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_q\}$ of subsets of C (i.e., a subset of 2^C), any of them verifying π and such that $\cup_{i=1}^{q} S_i = C$.

[3] A property is hereditary if whenever it is true for some set, it is true for any of its subsets; it is nontrivial if it is true for infinitely many sets and false for infinitely many sets also.

Then, MIN HEREDITARY COVER consists, given a property π, a ground set C and a family $\mathcal{S}$ including *any* subset of C verifying π, of determining a π-covering of minimum size. Observe that, by definition of the instances of MIN HEREDITARY COVER, singletons of the ground sets are included in any of them and are always sufficient to cover C. Henceforth, for any instance x of the problem, $\omega(x) = |C|$.

It is easy to see that, given a π-covering, one can yield a π-partition (i.e., a collection $\mathcal{S}$ where for any $S_i, S_j \in \mathcal{S}$, $S_i \cap S_j = \emptyset$) of the same size, by greedily removing duplications of elements of C. Henceforth, MIN HEREDITARY COVER or MIN HEREDITARY PARTITION are, in fact, the same problem. MIN HEREDITARY COVER has been introduced in Ref. [22] and revisited in Ref. [13] under the name MIN COVER BY INDEPENDENT SETS. Moreover, in the former paper, using a clever adaptation of the local improvement methods of Ref. [23], a differential ratio 3/4 for MIN HEREDITARY COVER has been proposed. Based on Ref. [24], this ratio has been carried to 289/360 by Ref. [13].

A lot of well-known **NPO** problems are instantiations of MIN HEREDITARY COVER. For instance, MIN COLORING becomes a MIN HEREDITARY COVER problem, considering as ground set the vertices of the input graph and as set system, the set of the independent sets[4] of this graph. The same holds for the partition of the covering of a graph by subgraphs that are planar, or by degree-bounded subgraphs, etc. Furthermore, if any element of C is associated with a weight and a subset S_i of C is in $\mathcal{S}$ if the total weight of its members is at most 1, then one recovers BIN PACKING.

In fact, an instance of MIN HEREDITARY COVER can be seen as a virtual instance of MIN SET COVER, even if there is no need to make it always explicit. Furthermore, the following general result links MIN k-SET COVER (the restriction of MIN SET COVER to subsets of cardinality at most k) and MIN HEREDITARY COVER (see Ref. [25] for its proof in the case of MIN COLORING; it can be easily seen that extension to the general MIN HEREDITARY COVER is immediate).

Theorem 16.1

If MIN k-SET COVER *is approximable in polynomial time within differential approximation ratio δ, then* MIN HEREDITARY COVER *is approximable in polynomial time within differential approximation ratio* $\min\{\delta, k/(k+1)\}$.

16.3.1.1 Min Coloring

MIN COLORING has been systematically studied in the differential paradigm. Subsequent papers [(18,23, 24,26–29)] have improved their differential approximation ratio from 1/2 to 289/360. This problem is also a typical example of a problem that behaves in completely different ways when dealing with the standard or the differential paradigms. Indeed, dealing with the former one, MIN COLORING is inapproximable within ratio $n^{1-\epsilon}$, for any $\epsilon > 0$, unless problems in **NP** can be solved by slightly superpolynomial deterministic algorithms (see Ref. [1] and Chapter 17).

As we have seen previously, given a graph $G(V, E)$, MIN COLORING can be seen as a MIN HEREDITARY COVER problem considering $C = V$ and taking for $\mathcal{S}$ the set of the independent sets of G. According to Theorem 16.1 and Ref. [24], where MIN 6-SET COVER is proved approximable within differential ratio 289/360, one can derive that it is also approximable within differential ratio 289/360. Note that any result for MIN COLORING also holds for the minimum vertex-partition (or covering) into cliques problem since an independent set in some graph G becomes a clique in the complement $\bar{G}$ of G (in other words, this problem is also an instantiation of MIN HEREDITARY COVER). Furthermore, in Refs. [26,27], a differential ratio preserving reduction is devised between minimum vertex-partition into cliques and minimum edge-partition (or covering) into cliques. So, as in the standard paradigm, all these three problems have identical differential approximation behavior.

Finally, it is proved in Ref. [30] that MIN COLORING is **DAPX**-complete (see also Section 16.5.3.1); consequently, unless **P** = **NP**, it cannot be solved by polynomial-time differential approximation schemata. This derives immediately that neither MIN HEREDITARY COVER belongs to **DPTAS**, unless **P** = **NP**.

[4] It is well known that the independence property is hereditary.

16.3.1.2 Bin Packing

We now deal with another very well-known **NPO** problem, the BIN PACKING. According to what has been discussed above, BIN PACKING being a particular case of MIN HEREDITARY COVER, it is approximable within differential ratio 289/360. In what follows in this section, we refine this result by first presenting an approximation preserving reduction transforming any standard approximation ratio ρ into differential approximation ratio $\delta = 2 - \rho$. Then, based on this reduction we show that BIN PACKING can be solved by a polynomial-time differential approximation schema [31]; in other words, BIN PACKING $\in$ **DPTAS**. This result draws another, although less dramatical than the one in Section 16.3.1.1, difference between standard and differential approximation. In the former paradigm, BIN PACKING is solved by an *asymptotic* polynomial-time approximation schema, more precisely within standard approximations ratio $1 + \epsilon + (1/\operatorname{opt}(L))$, for any $\epsilon > 0$ ([32]), but it is **NP**-hard to approximate it by a "real" polynomial-time approximation schema [2].

Consider a list $L = \{x_1, \ldots, x_n\}$, instance of BIN PACKING, assume, without loss of generality, that items in L are rational numbers ranged in decreasing order and fix an optimum solution B^* of L. Observe that $\omega(L) = n$. For the purposes of this section, a bin i will be denoted either by b_i, or by explicit listing of the numbers placed in it; finally, any solution will be alternatively represented as union of its bins.

Theorem 16.2

From any algorithm achieving standard approximation ratio ρ for BIN PACKING, *can be derived an algorithm achieving differential approximation ratio $\delta = 2 - \rho$.*

Proof (Sketch)

Let k^* be the number of bins in B^* that contain a single item. Then, it is easy to see that there exists an optimum solution $\bar{B}^* = \{x_1\} \cup \cdots \cup \{x_{k^*}\} \cup \bar{B}_2^*$ for L, where any bin in $\bar{B}_2^*$ contains at least two items. Furthermore, one can show that, for any optimum solution $\hat{B} = \{b_j : j = 1, \ldots, \operatorname{opt}(L)\}$ and for any set $J \subset \{1, \ldots, \operatorname{opt}(L)\}$, the solution $B_J = \{b_j \in B : j \in J\}$ is optimum for the sublist $L_J = \cup_{j \in J} b_j$.

Consider now Algorithm SA achieving standard approximation ratio ρ for BIN PACKING, denote by SA(L) the solution computed by it, when running on an instance L (recall that L is assumed ranged in decreasing order), and run the following algorithm, denoted by DA in the sequel, which uses SA as subprocedure:

1. for $k = 1$ to n set: $L_k = \{x_{k+1}, \ldots, x_n\}$, $B_k = \{x_1\} \cup \cdots \cup \{x_k\} \cup \mathtt{SA}(L_k)$;
2. output $B = \operatorname{argmin}\{|B_k| : k = 0, \ldots, n-1\}$.

Let $\bar{B}^*$ be the optimum solution claimed above. Then, $\bar{B}_2^*$ is an optimum solution for the sublist L_{k^*}. Observe that Algorithm SA called by DA has also been executed on L_{k^*} and denote by B_{k^*} the solution so computed by DA. The solution returned in step 2 verifies $|B| \leq |B_{k^*}|$. Finally, since any bin in $\bar{B}_2^*$ contains at least two items, $|L_{k^*}| = n - k^* \geq 2\operatorname{opt}(L_{k^*})$. Putting all this together, we get $\delta_{\mathtt{DA}}(L, B) = (n - |B|)/(n - \operatorname{opt}(L)) \geq (|L_{k^*}| - |B_{k^*}|)/(|L_{k^*}| - \operatorname{opt}(L_{k^*})) \geq 2 - \rho$. □

In what follows, denote by SA any polynomial algorithm approximately solving BIN PACKING within (fixed) constant standard approximation ratio ρ, by ASCHEMA(ϵ) the asymptotic polynomial-time standard approximation schema of Ref. [32], parameterized by $\epsilon > 0$, and consider the following algorithm, DSCHEMA (L is always assumed ranged in decreasing order):

1. fix a constant $\epsilon > 0$ and set $\eta = \lfloor 2(\rho - 1 + \epsilon)/\epsilon^2 \rfloor$;
2. for $k = n - \eta + 1, \ldots, n$ build list L_{k-1} where L_{k-1} is as in step 1 of Algorithm DA (Theorem 16.2);
3. for any list L_i computed in step 2 above, perform an exhaustive search on L_i, denote by E_i the solution so computed, and set $B_i = \{\{x\} : x \in L \setminus L_i\} \cup E_i$;
4. store B, the smallest of the solutions computed in Step 3;
5. run DA both with SA and ASCHEMA$(\epsilon/2)$, respectively, as subprocedures on L;
6. output the best among the three solutions computed in steps 4 and 5.

Theorem 16.3 (**Demange et al. [31]**)

Algorithm `DSCHEMA` *is a polynomial-time differential approximation schema for* BIN PACKING. *So,* BIN PACKING $\in$ ***DPTAS***.

Proof (Sketch)

Since ρ and ϵ do not depend on n, neither does η, computed at step 1. One can then show that when dealing with a list L such that $|L_{k^*+1}| \leq \eta$, BIN PACKING can be solved in polynomial time when η is a fixed constant. However, assuming $|L_{k^*+1}| \geq \eta$, then, one can prove that, if $\text{opt}(L_{k^*+1}) \leq \epsilon|L_{k^*+1}|/(\rho_{\texttt{SA}} - 1 + \epsilon)$, the approximation ratio of algorithm `DA`, when calling `SA` as subprocedure, is $\delta \geq 1 - \epsilon$ while, if $\text{opt}(L_{k^*+1}) \geq \epsilon|L_{k^*+1}|/(\rho_{\texttt{SA}} - 1 + \epsilon)$, then the approximation ratio of algorithm `DA`, when calling `ASCHEMA`$(\epsilon/2)$ as subprocedure, is also $\delta \geq 1 - \epsilon$. So, when $|L_{k^*+1}| \geq 2(\rho - 1 + \epsilon)/\epsilon^2$, step 5 of `DSCHEMA` achieves differential approximation ratio $1 - \epsilon$. Putting things together derives the result. □

Let us note that, as we will see in Section 16.5.4, BIN PACKING is **DPTAS**-complete; consequently, unless $\mathbf{P} = \mathbf{NP}$ it is inapproximable by fully polynomial-time differential approximation schemata. Inapproximability of BIN PACKING by such schemata has also been shown independently in Ref. [19].

16.3.2 Traveling Salesman Problems

MIN TSP is one of the most paradigmatic problems in combinatorial optimization and one of the hardest one to approximate. Indeed, unless $\mathbf{P} = \mathbf{NP}$, no polynomial algorithm can guarantee, on an edge-weighted complete graph of size n when no restriction is imposed to the edge weights, standard approximation ratio $O(2^{p(n)})$, for any polynomial p. As we will see in this section things are completely different when dealing with differential approximation where MIN TSP $\in$ **DAPX**. This result draws another notorious difference between the two paradigms.

Consider an edge-weighted complete graph of order n, denoted by K_n, and observe that the worst MIN TSP-solution in K_n is an optimum solution for MAX TSP. Consider the following algorithm (originally proposed by Monnot [33] for MAX TSP) based upon a careful patching of the cycles of a minimum-weight 2-matching[5] of K_n:

- compute $M = (C_1, C_2, \ldots, C_k)$; denote by $\{v_i^j : j = 1, \ldots, k, \ i = 1, \ldots, |C_j|\}$, the vertex set of C_j; if $k = 1$, return M;
- for any C_j, pick arbitrarily four consecutive vertices v_i^j, $i = 1, \ldots, 4$; if $|C_j| = 3$, $v_4^j = v_1^j$; for C_k (the last cycle of M), pick also another vertex, denoted by u that is the other neighbor of v_1^k in C_k (hence, if $|C_k| = 3$, then $u = v_3^k$ while if $|C_k| = 4$, then $u = v_4^k$);
- if k is even (odd), then set:

$$-R_1 = \cup_{j=1}^{k-1}\{(v_2^j, v_3^j)\} \cup \{(v_1^k, v_2^k)\},\ A_1 = \{(v_1^k, v_3^1), (v_2^1, v_2^2)\} \cup_{j=1}^{(k-2)/2} \{(v_3^{2j}, v_3^{2j+1}), (v_2^{2j+1}, v_2^{2j+2})\}$$

$$(R_1 = \cup_{j=1}^{k}\{(v_2^j, v_3^j)\},\ A_1 = \{(v_2^k, v_3^1)\} \cup_{j=1}^{(k-1)/2} \{(v_2^{2j-1}, v_2^{2j}), (v_3^{2j}, v_3^{2j+1})\}),\ T_1 = (M \setminus R_1) \cup A_1;$$

$$-R_2 = \cup_{j=1}^{k-1}\{(v_1^j, v_2^j)\} \cup \{(u, v_1^k)\},\ A_2 = \{(u, v_2^1), (v_1^1, v_1^2)\} \cup_{j=1}^{(k-2)/2} \{(v_2^{2j}, v_2^{2j+1}), (v_1^{2j+1}, v_1^{2j+2})\}$$

$$(R_2 = \cup_{j=1}^{k}\{(v_1^j, v_2^j)\},\ A_2 = \{(v_1^k, v_2^1)\} \cup_{j=1}^{(k-1)/2} \{(v_1^{2j-1}, v_1^{2j}), (v_2^{2j}, v_2^{2j+1})\}),\ T_2 = (M \setminus R_2) \cup A_2;$$

$$-R_3 = \cup_{j=1}^{k-1}\{(v_3^j, v_4^j)\} \cup \{(v_2^k, v_3^k)\},\ A_3 = \{(v_2^k, v_4^1), (v_3^1, v_3^2)\} \cup_{j=1}^{(k-2)/2} \{(v_4^{2j}, v_4^{2j+1}), (v_3^{2j+1}, v_3^{2j+2})\}$$

$$(R_3 = \cup_{j=1}^{k}\{(v_3^j, v_4^j)\},\ A_3 = \{(v_3^k, v_4^1)\} \cup_{j=1}^{(k-1)/2} \{(v_3^{2j-1}, v_3^{2j}), (v_4^{2j}, v_4^{2j+1})\}),\ T_3 = (M \setminus R_3) \cup A_3;$$

- output T the best among T_1, T_2, and T_3.

[5] A minimum-weight 2-matching is a minimum total weight partial subgraph of K_n any vertex of which has degree at most 2; this computation is polynomial, see, for example, Ref. [34]; in other words, a 2-matching is a collection of paths and cycles, but when dealing with complete graphs a 2-matching can be considered as a collection of cycles.

As proved in Ref. [33], the set $(M \setminus \cup_{i+1}^{3} R_i) \cup_{i+1}^{3} A_i$ is a feasible solution for MIN TSP, the value of which is a lower bound for $\omega(K_n)$; furthermore, $m(K_n, T) \leq (\sum_{i=1}^{3} m(K_n, T_i))/3$. Then, a smart analysis, leads to the following theorem (the same result has been obtained, by a different algorithm working also for negative edge weights, in Ref. [13]).

Theorem 16.4 (Monnot [33])

MIN TSP *is differentially 2/3-approximable.*

Notice that MIN TSP, MAX TSP, MIN METRIC TSP, and MAX METRIC TSP are all affine equivalent (see Ref. [35] for the proof; for the two former problems, just replace weight $d(i, j)$ of edge (v_i, v_j) by $M - d(i, j)$, where M is some number greater than the maximum edge weight). Hence, the following theorem holds.

Theorem 16.5

MIN TSP, MAX TSP, MIN METRIC TSP, *and* MAX METRIC TSP *are differentially 2/3-approximable.*

A very famous restrictive version of MIN METRIC TSP is the MIN TSP12, where edge weights are all either 1 or 2. In Ref. [36], it is proved that this version (as well as, obviously, MAX TSP 12) is approximable within differential ratio 3/4.

16.3.3 Min Multiprocessor Scheduling

We now deal with a classical scheduling problem, the MIN MULTIPROCESSOR SCHEDULING [37], where we are given n tasks $t_1, \ldots, t_n$ with (execution) time lengths $l(t_j)$, $j = 1, \ldots, n$, polynomial with n, that have to be executed on m processors, and the objective is to partition these tasks on the processors in such a way that the occupancy of the busiest processor is minimized. Observe that the worst solution is the one where all the tasks are executed in the same processor; so, given an instance x of MIN MULTIPROCESSOR SCHEDULING, $\omega(x) = \sum_{j=1}^{n} l(t_j)$. A solution y of this problem will be represented as a vector in $\{0, 1\}^{mn}$, the nonzero components y_j^i of which correspond to the assignment of task j to processor i.

Consider a simple local search algorithm that starts from some solution and improves it upon any change of the assignment of a single task from one processor to another. Then the following result can be obtained [38].

Theorem 16.6

MIN MULTIPROCESSOR SCHEDULING *is approximable within differential ratio $m/(m+1)$.*

Proof (Sketch)

Assume that both tasks and processors are ranged with decreasing lengths and occupancies, respectively. Denote by $l(p_i)$, the total occupancy of processor p_i, $i = 1, \ldots, m$. Then, $\text{opt}(x) \geq l(t_1)$ and $l(p_1) = \sum_{j=1}^{n} y_j^1 l(t_j) = \max_{i=1,\ldots,} m\{l(p_i) = \sum_{j=1}^{n} y_j^i l(t_j)\}$. Denote, w.l.o.g., by $1, \ldots, q$, the indices of the tasks assigned to p_1. Since y is a local optimum, it verifies, for $i = 2, \ldots, m$, $j = 1, \ldots, q$: $l(t_j) + l(p_i) \geq l(p_1)$. We can assume $q \geq 2$ (on the contrary y is optimum). Then, adding the preceding expression for $j = 1, \ldots, q$, we get $l(p_i) \geq l(p_1)/2$. Also, adding $l(p_1)$ with the preceding expression for $l(p_i)$, $i = 2, \ldots, m$, we obtain $\omega(x) \geq (m+1)l(p_1)/2$. Putting all this together we finally get $m(x, y) = l(p_1) \leq (m\,\text{opt}(x)/(m+1)) + (\omega(x)/(m+1))$. □

16.4 Asymptotic Differential Approximation Ratio

In any approximation paradigm, the notion of asymptotic approximation (dealing, informally, with a class of "interesting" instances) is pertinent. In the standard paradigm, the asymptotic approximation ratio is defined on the hypothesis that the interesting (from an approximation point of view) instances of the simple problems are the ones whose values of the optimum solutions tend to ∞ (because, in the opposite case,[6] these problems, called *simple* [39], are polynomial). In the differential approximation framework, on the contrary, the size (or the value) of the optimum solution is not always a pertinent hardness criterion (see Ref. [40] for several examples about this claim). Henceforth, in Ref. [40], another hardness criterion, *the number* $\sigma(x)$ *of the feasible values of* x, has been used to introduce the *asymptotic differential approximation ratio.* Under this criterion, the asymptotic differential approximation ratio of an algorithm $\mathtt{A}$ is defined as

$$\delta_{\mathtt{A}}^{\infty}(x, y) = \lim_{k \to \infty} \inf_{\substack{x \\ \sigma(x) \geq k}} \left\{ \frac{\omega(x) - m(x, y)}{\omega(x) - \mathrm{opt}(I)} \right\} \tag{16.1}$$

Let us note that $\sigma(x)$ is motivated by, and generalizes, the notion of the *structure of the instance* introduced in Ref. [9]. We also notice that the condition $\sigma(x) \geq k$ characterizing "the sequence of unbounded instances" (or "limit instances") cannot be polynomially verified.[7] But in practice, for a given problem, it is possible to directly interpret condition $\sigma(x) \geq k$ by means of the parameters $\omega(x)$ and opt(x) (note that $\sigma(x)$ is not a function of these values). For example, for numerous cases of discrete problems, it is possible to determine, for any instance x, a step $\pi(x)$ defined as the least variation between two feasible values of x. For example, for BIN PACKING, $\pi(x) = 1$. Then, $\sigma(x) \leq ((\omega(x) - \mathrm{opt}(x))/\pi(x)) + 1$. Therefore, from Eq. (16.1):

$$\delta_{\mathtt{A}}^{\infty}(x, y) \geq \lim_{k \to \infty} \inf_{\substack{x \\ \frac{\omega(x) - \mathrm{opt}(x)}{\pi(x)} \geq k-1}} \left\{ \frac{\omega(x) - m(x, y)}{\omega(x) - \mathrm{opt}(x)} \right\}$$

Whenever π can be determined, condition $(\omega(x) - \mathrm{opt}(x))/\pi(x) \geq k - 1$ can be easier to evaluate than $\sigma(x) \geq k$, and in this case, the former condition is used (this is not senseless since we try to bound below the ratio).

The adoption of $\sigma(x)$ as hardness criterion can be motivated by considering a class of problems, called *radial problems* in Ref. [40], that includes many well-known combinatorial optimization problems, as BIN PACKING, MAX INDEPENDENT SET, MIN VERTEX COVER, MIN COLORING, etc. Informally, a problem Π is radial if, given an instance x of Π and a feasible solution y for x, one can, in polynomial time, on the one hand, deteriorate y as much as one wants (up to finally obtain a worst-value solution) and, on the other, greedily improve y to obtain (always in polynomial time) a suboptimal solution (eventually the optimum one).

Definition 16.2

A problem $\Pi = (\mathcal{I}, \mathrm{Sol}, m, \mathrm{goal})$ *is radial if there exists three polynomial algorithms* ξ, ψ, *and* ϕ *such that, for any* $x \in \mathcal{I}$:

1. ξ *computes a feasible solution* $y^{(0)}$ *for* x;
2. *for any feasible solution* y *of* x *strictly better (in the sense of the value) than* $y^{(0)}$, *algorithm* ϕ *computes a feasible solution* $\phi(y)$ *(if any) with* $m(x, \phi(y))$ *strictly worse than* $m(x, y)$;
3. *for any feasible solution* y *of* x *with value strictly better than* $m(x, y^{(0)})$, *there exists* $k \in \mathbb{N}$ *such that* $\phi^k(y) = y^{(0)}$ *(where* ϕ^k *denotes the k-times iteration of* ϕ*);*
4. *for a solution* y *such that, either* $y = y^{(0)}$, *or* y *is any feasible solution of* x *with value strictly better than* $m(x, y^{(0)})$, $\psi(y)$ *computes the set of ancestors of* y, *defined by* $\psi(y) = \phi^{-1}(\{y\}) = \{z : \phi(z) = y\}$ *(this set being eventually empty).*

Let us note that the class of radial problems includes in particular the well-known class of *hereditary* problems for which any subset of a feasible solution remains feasible. In fact, for a hereditary (maximization) problem, a feasible solution y is a subset of the input data, for any instances x, $y^{(0)} = \emptyset$, and for any other feasible solution y, $\phi(y)$ is just obtained from y by removing a component of y. The hereditary notion deals with problems for which a feasible solution is a subset of the input data, while the radial notion allows problems for which solutions are also second-order structures of the input data.

[6] The case where optimum values are bounded by fixed constants.

[7] The same holds for the condition opt$(x) \geq k$ induced by the hardness criterion in the standard paradigm.

Proposition 16.1 **(Demange and Paschos [40])**

Let κ be a fixed constant and consider a radial problem Π such that, for any instance x of Π of size n, $\sigma(x) \leq \kappa$. Then, Π is polynomial-time solvable.

16.5 Structure in Differential Approximation Classes

What has been discussed in the previous sections makes it clear which the entire theory of approximation, which tries characterize and classify problems with respect to their approximability hardness, can be redone in the differential paradigm. There exist problems having several differential approximability levels and inapproximability bounds. What follows further confirms this claim. It will be shown that the approximation paradigm we deal with allows to devise its proper tools and to use them to design an entire structure for the approximability classes involved.

16.5.1 Differential NPO-Completeness

Obviously, the *strict* reduction of Ref. [41] (see also Chapter 15), can be identically defined in the framework of the differential approximation; for clarity, we denote this derivation of the strict reduction by D-reduction. Two **NPO** problems will be called D-equivalent if there exist D-reductions from any of them to the other one.

Theorem 3.1 in Ref. [41] (where the differential approximation ratio is mentioned as a possible way of estimating the performance of an algorithm), based upon an extension of Cook's proof [42] of SAT **NP**-completeness to optimization problems, works also when the differential ratio is dealt instead of the standard one. Furthermore, solution *triv*, as defined in Ref. [41] is indeed a worst solution for MIN WSAT. However, the following proposition holds.

Proposition 16.1 **(Ausiello et al. [43])**

MAX WSAT *and* MIN WSAT *are D-equivalent.*

Proof (Sketch)

With any clause $\ell_1 \vee \cdots \vee \ell_t$ of an instance ϕ of MAX WSAT, we associate in the instance ϕ' of MIN WSAT the clause $\bar{\ell}_1 \vee \cdots \vee \bar{\ell}_t$. Then, if an assignment y satisfies the instance ϕ, the complement y' of y satisfies ϕ', and vice versa. So, $m(\phi, y) = \sum_{i=1}^{n} w(x_i) - m(\phi', y')$, for any y'. Thus, $\delta(\phi, y) = \delta(\phi', y')$. The reduction from MIN WSAT to MAX WSAT is completely analogous. □

In a completely analogous way, as in Proposition 16.1, it can be proved that MIN 0-1 INTEGER PROGRAMMING and MAX 0-1 INTEGER PROGRAMMING are also D-equivalent. Putting all the above together the following holds.

Theorem 16.7

MAX WSAT, MIN WSAT, MIN 0-1 INTEGER PROGRAMMING, *and* MAX 0-1 INTEGER PROGRAMMING *are **NPO**-complete under D-reducibility.*

16.5.2 The Class 0-DAPX

Informally, the class 0-DAPX is the class of NPO problems for which the differential ratio of any polynomial-time algorithm is *equal* to 0. In other words, for any such algorithm, there exists an instance on which it will compute its worst solution. Such situation draws the worst case for the differential approximability of a problem. Class 0-DAPX is defined in Ref. [43] by means of a reduction called G-reduction. It can be seen as a particular kind of the GAP-reduction [1,44,45].

Definition 16.3

A problem Π is said to be G-reducible to a problem Π', if there exists a polynomial reduction that transforms any δ-differential approximation algorithm for Π', $\delta > 0$, into an optimum (exact) algorithm for Π.

Let Π be an **NP**-complete decision problem and Π' an **NPO** problem. The underlying idea for $\Pi \leq_{\mathrm{G}} \Pi'$ in Definition 16.3 is, starting from an instance of Π, to construct instances for Π' that have only two distinct feasible values and to prove that any differential δ-approximation for Π', $\delta > 0$, could distinguish between positive instances and negative instances for Π. Note finally that the G-reduction generalizes both the D-reduction of Section 16.5.1 and the strict reduction of Ref. [41].

Definition 16.4

***0-DAPX** is the class of **NPO** problems Π' for which there exists an **NP**-complete problem Π G-reducible to Π'. A problem is said to **0-DAPX**-hard, if any problem in **0-DAPX** G reduces to it.*

An obvious consequence of Definition 16.4 is that ***0-DAPX** is the class of **NPO** problems Π for which approximation within any differential approximation ratio $\delta > 0$ would entail $\mathbf{P} = \mathbf{NP}$.*

Proposition 16.3 (Bazgan and Paschos [46])

MIN INDEPENDENT DOMINATING SET $\in$ ***0-DAPX**.*

Proof (Sketch)

Given an instance ϕ of SAT with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$, construct a graph G, instance of MIN INDEPENDENT DOMINATING SET associating with any positive literal x_i a vertex u_i and with any negative literal $\bar{x}_i$ a vertex v_i. For $i = 1, \ldots, n$, draw edges (u_i, v_i). For any clause C_j, add in G a vertex w_j and an edge between w_j and any vertex corresponding to a literal contained in C_j. Finally, add edges in G to obtain a complete graph on $w_1, \ldots, w_m$. An independent set of G contains at most $n + 1$ vertices. An independent dominating set containing the vertices corresponding to true literals of a nonsatisfiable assignment and one vertex corresponding to a clause not satisfied by this assignment, is a worst solution of G of size $n + 1$. If ϕ is satisfiable then $\mathrm{opt}(G) = n$. If ϕ is not satisfiable then $\mathrm{opt}(G) = n + 1$. So, any independent dominating set of G has cardinality either n or $n + 1$. □

By analogous reductions, restricted versions of optimum-weighted satisfiability problems are proved **0-DAPX** in Ref. [47].

Finally, the following relationship between **NPO** and **0-DAPX** holds.

Theorem 16.8 (Ausiello et al. [43])

*Under D-reducibility, **NPO**-complete = **0-DAPX**-complete $\subseteq$ **0-DAPX**.*

If, instead of D, a stronger reducibility is considered, for instance, by allowing f and/or g to be multivalued in the strict reduction, then, under this type of reducibility, it can be proved that **NPO**-complete = **0-DAPX** [43].

16.5.3 DAPX- and Poly-DAPX-Completeness

In this section we address the problem of completeness in the classes **DAPX** and **Poly-DAPX**. For this purpose, we first introduce a differential approximation schemata preserving reducibility, originally presented in Ref. [43], called DPTAS-reducibility.

Definition 16.5

*Given two **NPO** problems Π and Π', Π DPTAS reduces to Π' if there exist a (possibly) multivalued function $f = (f_1, f_2, \ldots, f_h)$, where h is bounded by a polynomial in the input length, and two functions g and c, computable in polynomial time, such that*

- *for any $x \in \mathcal{I}_\Pi$, for any $\epsilon \in (0, 1) \cap \mathbb{Q}$, $f(x, \epsilon) \subseteq \mathcal{I}_{\Pi'}$;*
- *for any $x \in \mathcal{I}_\Pi$, for any $\epsilon \in (0, 1) \cap \mathbb{Q}$, for any $x' \in f(x, \epsilon)$, for any $y \in \mathrm{sol}_{\Pi'}(x')$, $g(x, y, \epsilon) \in \mathrm{sol}_\Pi(x)$;*
- *$c : (0, 1) \cap \mathbb{Q} \rightarrow (0, 1) \cap \mathbb{Q}$;*
- *for any $x \in \mathcal{I}_\Pi$, for any $\epsilon \in (0, 1) \cap \mathbb{Q}$, for any $y \in \cup_{i=1}^{h} \mathrm{sol}_{\Pi'}(f_i(x, \epsilon))$, $\exists j \leqslant h$ such that $\delta_{\Pi'}(f_j(x, \epsilon), y) \geqslant 1 - c(\epsilon)$ implies $\delta_\Pi(x, g(x, y, \epsilon)) \geqslant 1 - \epsilon$.*

16.5.3.1 DAPX-Completeness

If one restricts her/himself to problems with polynomially computable worst solutions, then things are rather simple. Indeed, given such a problem $\Pi \in$ **DAPX**, it is affine equivalent to a problem Π' defined on the same set of instances and with the same set of solutions but, for any solution y of an instance x of Π, the measure for solution y with respect to Π' is defined as $m_{\Pi'}(x, y) = m_{\Pi}(x, y) - \omega(x)$. Affine equivalence of Π and Π' ensures that $\Pi' \in$ **DAPX**; furthermore, $\omega_{\Pi'}(x) = 0$. Since, for the latter problem, standard and differential approximation ratios coincide, it follows that $\Pi' \in$ **APX**. MAX INDEPENDENT SET is **APX**-complete under PTAS-reducibility [48], a particular kind of the AP-reducibility seen in Chapter 15. So, Π' PTAS reduces to MAX INDEPENDENT SET. Putting together affine equivalence between Π and Π', PTAS-reducibility between Π' and MAX INDEPENDENT SET, and taking into account that composition of these two reductions is an instantiation of DPTAS-reduction, we conclude the **DAPX**-completeness of MAX INDEPENDENT SET.

However, things become much more complicated, if one takes into account problems with nonpolynomially computable worst solutions. In this case, one needs more sophisticated techniques and arguments. We informally describe here the basic ideas and the proof schema in Ref. [43]. It is first shown that any DAPX problem Π is reducible to MAX WSAT-B by a reduction transforming a polynomial-time approximations schema for MAX WSAT-B into a polynomial-time differential approximation schema for Π. For simplicity, denote this reduction by S–D. Next, a particular **APX**-complete problem Π' is considered, say MAX INDEPENDENT SET-B. MAX WSAT-B, that is in **APX**, is PTAS-reducible to MAX INDEPENDENT SET-B. MAX INDEPENDENT SET-B is both in **APX** and in **DAPX** and, moreover, standard and differential approximation ratios coincide for it; this coincidence draws a trivial reduction called ID-reduction. It trivially transforms a differential polynomial-time approximation schema into a standard polynomial-time approximation schema. The composition of the three reductions specified (i.e., the S–D-reduction from Π to MAX WSAT-B, the PTAS-reduction from MAX WSAT-B to MAX INDEPENDENT SET-B, and the ID-reduction) is a DPTAS-reduction transforming a polynomial-time differential approximation schema for MAX INDEPENDENT SET-B into a polynomial-time differential approximation schema for Π, i.e., MAX INDEPENDENT SET-B is **DAPX**-complete under DPTAS-reducibility.

Also, by standard reductions that turn out to be DPTAS-reductions also, the following can be proved [30,43].

Theorem 16.9

MAX INDEPENDENT SET-B, MIN VERTEX COVER-B, *for fixed* B, MAX k-SET PACKING, MIN k-SET COVER, *for fixed k, and* MIN COLORING *are **DAPX**-complete under DPTAS-reducibility.*

16.5.3.2 Poly-DAPX-Completeness

Recall that a maximization problem $\Pi \in$ **NPO** is *canonically hard for **Poly-APX*** [49], if and only if there exist a polynomially computable transformation T from 3SAT to Π, two constants n_0 and c and a function F, hard for **Poly**,[8] such that, given an instance x of 3SAT on $n \geqslant n_0$ variables and a number $N \geqslant n^c$, the instance $x' = T(x, N)$ belongs to $\mathcal{I}_{\Pi}$ and verifies the following three properties: (i) if x is satisfiable, then $\text{opt}(x') = N$; (ii) if x is not satisfiable, then $\text{opt}(x') = N/F(N)$; (iii) given a solution $y \in \text{sol}_{\Pi}(x')$ such that $m(x', y) > N/F(N)$, one can polynomially determine a truth assignment satisfying x.

Based on DPTAS-reducibility and the notion of canonical hardness, the following is proved in Ref. [30].

Theorem 16.10

*If a (maximization) problem $\Pi \in$ **NPO** is canonically hard for **Poly-APX**, then any problem in **Poly-DAPX** DPTAS reduces to Π.*

[8] The set of functions from $\mathbb{N}$ to $\mathbb{N}$ is bounded by a polynomial; a function $f \in$ **Poly** is hard for Poly, if and only if there exists three constants k, c, and n_0 such that, for any $n \geqslant n_0$, $f(n) \leqslant kF(n^c)$.

As it is shown in Ref. [49], MAX INDEPENDENT SET is canonically hard for **Poly-APX**. Furthermore, MIN VERTEX COVER is affine equivalent to MAX INDEPENDENT SET. Henceforth, use of Theorem 16.10 immediately derives the following result.

Theorem 16.11

MAX INDEPENDENT SET *and* MIN VERTEX COVER *are complete for* ***Poly-DAPX*** *under DPTAS-reducibility.*

16.5.4 DPTAS-Completeness

Completeness in **DPTAS** (the class of **NPO** problems that are approximable by polynomial time differential approximation schemata) is tackled by means of a kind of reducibility preserving membership in **DFPTAS**, which is called DFT-reducibility in Ref. [30]. This type of reducibility is the differential counterpart of the FT-reducibility introduced in Section 15.7 of Chapter 15 and can be defined in an exactly similar way. Based on DFT-reducibility, the following theorem holds ([30]; its proof is very similar to the one of Theorem 15.4 in Chapter 15). Before stating it, we need to introduce the class of diameter polynomially bounded problems that is a subclass of the radial problems seen in Section 16.4. An **NPO** problem Π is *diameter polynomially bounded* if and only if, for any $x \in \mathcal{I}_{\Pi}$, $|\operatorname{opt}(x) - \omega(x)| \leqslant q(|x|)$. The class of diameter polynomially bounded **NPO** problems will be denoted by **NPO-DPB**.

Theorem 16.12 (Bazgan et al. [30])

*Let Π' be an **NP**-hard problem **NPO-DPB**. Then, any problem in **NPO** is DFT reducible to Π'. Consequently, (i) the closure of **DPTAS** under DFT-reductions is the whole **NPO** and (ii) any **NP**-hard problem in **NPO-DPB** $\cap$ **DPTAS** is **DPTAS**-complete under DFT-reductions.*

Consider now MIN PLANAR VERTEX COVER, MAX PLANAR INDEPENDENT SET, and BIN PACKING. They are all **NP**-hard and in **NPO-DPB**. Furthermore, they are all in **DPTAS** (for the first two problems, this is derived by the inclusion of MAX PLANAR INDEPENDENT SET in **PTAS** proved in Ref. [50]; for the third one, see Section 16.3.1.2). So, the following theorem holds and concludes this section [30].

Theorem 16.13

MAX PLANAR INDEPENDENT SET, MIN PLANAR VERTEX COVER, *and* BIN PACKING *are* ***DPTAS****-complete under DFT-reducibility.*

16.6 Discussion and Final Remarks

As we have already claimed in the beginning of Section 16.5, the entire theory of approximation can be reformulated in the differential paradigm. This paradigm has the diversity of the standard one, it has a nonempty scientific content and, to our opinion, it represents in some sense a kind of revival for the domain of the polynomial approximation.

Since the work in Ref. [7], a great number of paradigmatic combinatorial optimization problems has been studied in the framework of the differential approximation. For instance, KNAPSACK has been studied in Ref. [7] and revisited in and Ref. [13]. MAX CUT, MIN CLUSTER, STACKER CRANE, MIN DOMINATING SET, MIN DISJOINT CYCLE COVER, and MAX ACYCLIC SUBGRAPH have been dealt in Ref. [13]. MIN FEEDBACK ARC SET is also studied in Ref. [38] together with MIN FEEDBACK NODE SET. MIN VERTEX COVER and MAX INDEPENDENT SET are studied in Refs. [7,13]. MIN COLORING is dealt in Ref. [18,23,24, 26–30], while MIN WEIGHTED COLORING (where the input is a vertex-weighted graph and the weight of a color is the weight of the heaviest of its vertices) is studied in Ref. [51] (see also Ref. [52]). MIN INDEPENDENT DOMINATING SET is dealt in Ref. [46]. BIN PACKING is studied in Refs. [27,31,40,53]. MIN SET COVER, under several assumptions on its worst value, is dealt in Refs. [7,13,54], while MIN WEIGHTED SET COVER is dealt in Refs. [27,54]. MIN TSP and MAX TSP, as well as, several famous variants of them, MIN METRIC TSP, MAX METRIC TSP, MIN TSPab (the most famous restrictive case of this problem is MIN TSP12), and MAX TSPab are studied in Refs. [13,33,35,36,55]. STEINER TREE problems under several assumptions on the form of the input graph and on the edge weights are dealt in Ref. [56]. Finally, several optimum satisfiability and constraint satisfaction problems (as MAX SAT, MAX E2SAT, MAX 3SAT, MAX E3SAT, MAX EkSAT, MIN SAT, MIN kSAT, MIN EkSAT, MIN 2SAT, and their corresponding constraint satisfaction versions) are studied in Ref. [57].

Dealing with structural aspects of approximation, besides the existing approximability classes (defined rather upon combinatorial arguments) two logical classes have been very notorious in the standard paradigm. These are **Max-NP** and **Max-SNP**, originally introduced in Ref. [58] (see also Chapters 15 and 17). Their definitions, independent from any approximation ratio consideration, make that they can identically be considered also in differential approximation. In the standard paradigm, the following strict inclusions hold: **PTAS** $\subset$ **Max-SNP** $\subset$ **APX** and **MAX-NP** $\subset$ **APX**. As it is proved in Ref. [57], MAX SAT $\notin$ **DAPX**, unless **P** $=$ **NP**. This, draws an important structural difference in the landscape of approximation classes in the two paradigms, since an immediate corollary of this result is that **MAX-NP** $\not\subset$ **DAPX**. Position of **Max-SNP** in the differential landscape is not known yet. It is conjectured, however, that **MAX-SNP** $\not\subset$ **DAPX**. In any case, formal relationships of **Max-SNP** and **Max-NP** with the other differential approximability classes deserve further study.

References

[1] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M., *Complexity and Approximation. Combinatorial Optimization Problems and Their Approximability Properties*, Springer, Berlin, 1999.

[2] Garey, M. R. and Johnson, D. S., *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[3] Berge, C., *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.

[4] Bollobás, B., *Random Graphs*, Academic Press, London, 1985.

[5] Sahni, S. and Gonzalez, T., P-complete approximation problems, *JACM*, 23, 555, 1976.

[6] Håstad, J., Clique is hard to approximate within $n^{1-\epsilon}$, *Acta Mathematica*, 182, 105, 1999.

[7] Demange, D. and Paschos, V. Th., On an approximation measure founded on the links between optimization and polynomial approximation theory, *Theor. Comput. Sci.*, 158, 117, 1996.

[8] Ausiello, G., D'Atri, A., and Protasi, M., On the structure of combinatorial problems and structure preserving reductions, *Proc. ICALP'77*, Lecture Notes in Computer Science, Vol. 52, Springer, Berlin, 1977, p. 45

[9] Ausiello, G., D'Atri, A., and Protasi, M., Structure preserving reductions among convex optimization problems, *JCSS*, 21, 136, 1980.

[10] Aiello, A., Burattini, E., Furnari, M., Massarotti, A., and Ventriglia, F., Computational complexity: the problem of approximation, in *Algebra, Combinatorics, and Logic in Computer Science*, Vol. I, Colloquia Mathematica Societatis, János Bolyai, North-Holland, New York, 1986, p. 51.

[11] Zemel, E., Measuring the quality of approximate solutions to zero–one programming problems, *Math. Oper. Res.*, 6, 319, 1981.

[12] Cornuejols, G., Fisher, M. L., and Nemhauser, G. L., Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms, *Manag. Sci.*, 23(8), 789, 1977.

[13] Hassin, R. and Khuller, S., z-Approximations, *J. Algorithms*, 41, 429, 2001.

[14] Bellare, M. and Rogaway, P., The complexity of approximating a nonlinear program, *Math. Program.*, 69, 429, 1995.

[15] Nemirovski, A. S. and Yudin, D. B., *Problem Complexity and Method Efficiency in Optimization*, Wiley, Chichester, 1983.

[16] Vavasis, S. A., Approximation algorithms for indefinite quadratic programming, *Math. Program.*, 57, 279, 1992.

[17] Ausiello, G., D'Atri, A., and Protasi, M., Lattice-theoretical ordering properties for NP-complete optimization problems, *Fundamenta Informaticæ*, 4, 83, 1981.

[18] Hassin, R. and Lahav, S., Maximizing the number of unused colors in the vertex coloring problem, *Inform. Process. Lett.*, 52, 87, 1994.

[19] Demange, M. and Paschos, V. Th., Improved approximations for maximum independent set via approximation chains, *Appl. Math. Lett.*, 10(3), 105, 1997.

[20] Demange, M. and Paschos, V. Th., Improved approximations for weighted and unweighted graph problems, *Theor. Comput. Syst.*, to appear, preliminary version available at `http://l1.lamsade.dauphine.fr/~paschos/documents/c177.pdf`.

[21] Halldórsson, M. M., Approximations of weighted independent set and hereditary subset problems, *J. Graph Algorithms Appl.*, 4(1), 1, 2000.

[22] Monnot, J., Critical Families of Instances and Polynomial Approximation. Ph.D. thesis, LAMSADE, University Paris-Dauphine, 1998 (in French).

[23] Halldórsson, M. M., Approximating k-set cover and complementary graph coloring, *Proc. Int. Integer Programming and Combinatorial Optimization Conf.*, Lecture Notes in Computer Science, Vol. 1084, Springer, Berlin, 1996, p. 118.

[24] Duh, R. and Frer, M., Approximation of k-set cover by semi-local optimization, *Proc. of STOC*, 1997, p. 256.

[25] Paschos, V. Th., Polynomial approximation and graph coloring, *Computing*, 70, 41, 2003.

[26] Demange, M., Grisoni, P., and Paschos, V. Th., Approximation results for the minimum graph coloring problem, *Inform. Process. Lett.*, 50, 19, 1994.

[27] Demange, M., Grisoni, P., and Paschos, V. Th., Differential approximation algorithms for some combinatorial optimization problems, *Theor. Comput. Sci.*, 209, 107, 1998.

[28] Halldórsson, M. M., Approximating discrete collections via local improvements, *Proc. of SODA*, 1995, p. 160.

[29] Tzeng, X. D. and King, G. H., Three-quarter approximation for the number of unused colors in graph coloring, *Inform. Sci.*, 114, 105, 1999.

[30] Bazgan, C., Escoffier, B., and Paschos, V. Th., Completeness in standard and differential approximation classes: poly-(D)APX- and (D)PTAS-completeness, *Theor. Comput. Sci.*, 339, 272, 2005.

[31] Demange, M., Monnot, J., and Paschos, V. Th., Bridging gap between standard and differential polynomial approximation: the case of bin-packing, *Appl. Math. Lett.*, 12, 127, 1999.

[32] Fernandez de la Vega, W. and Lueker, G. S., Bin packing can be solved within $1 + \epsilon$ in linear time, *Combinatorica*, 1(4), 349, 1981.

[33] Monnot, J., Differential approximation results for the traveling salesman and related problems, *Inform. Process. Lett.*, 82(5), 229, 2002.

[34] Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Schrijver, A., *Combinatorial Optimization*, Wiley, New York, 1998.

[35] Monnot, J., Paschos, V. Th., and Toulouse, S., Approximation algorithms for the traveling salesman problem, *Math. Methods Oper. Res.*, 57(1), 387, 2003.

[36] Monnot, J., Paschos, V. Th., and Toulouse, S., Differential approximation results for the traveling salesman problem with distances 1 and 2, *Eur. J. Oper. Res.*, 145(3), 557, 2002.

[37] Hochbaum, D. S. and Shmoys, D. B., Using dual approximation algorithms for scheduling problems: theoretical and practical results, *JACM*, 34, 144, 1987.

[38] J. Monnot, J., Paschos, V. Th., and Toulouse, S., Optima locaux garantis pour l'approximation diffrentielle, *Tech. et Sci. Informatiques*, 22(3), 257, 2003.

[39] Paz, A. and Moran, S., Non deterministic polynomial optimization problems and their approximations, *Theor. Comput. Sci.*, 15, 251, 1981.

[40] Demange, M. and Paschos, V. Th., Asymptotic differential approximation ratio: definitions, motivations and application to some combinatorial problems, *RAIRO Oper. Res.*, 33, 481, 1999.

[41] Orponen, P. and Mannila, H., On Approximation Preserving Reductions: Complete Problems and Robust Measures, Technical report C-1987-28, Department of CS, University of Helsinki, Finland, 1987.

[42] Cook, S. A., The complexity of theorem-proving procedures, *Proc. of STOC'71*, 1971, p. 151.

[43] Ausiello, G., Bazgan, C., Demange, M., and Paschos, V. Th., Completeness in differential approximation classes, *Int. J. Found. Comput. Sci.*, 16(6), 1267, 2005.

[44] Arora, S. and Lund, C., Hardness of approximation, in *Approximation Algorithms for NP-Hard Problems*, Hochbaum, D. S., Ed., PWS, Boston, 1997, chap. 10.

[45] Vazirani, V., *Approximation Algorithms*, Springer, Berlin, 2001.

[46] Bazgan, C. and Paschos, V. Th., Differential approximation for optimal satisfiability and related problems, *Eur. J. Oper. Res.*, 147(2), 397, 2003.

[47] Paschos, V. Th., *Complexit et Approximation Polynomiale*, Herms, Paris, 2004.

[48] Crescenzi, P. and Trevisan, L., On approximation scheme preserving reducibility and its applications, in *Foundations of Software Technology and Theoretical Computer Science, FST-TCS*, Lecture Notes in Computer Science, Vol. 880, Springer, Berlin, 1994, p. 330.

[49] Khanna, S., Motwani, R., Sudan, M., and Vazirani, U., On syntactic versus computational views of approximability, *SIAM J. Comput.*, 28, 164, 1998.

[50] Baker, B. S., Approximation algorithms for NP-complete problems on planar graphs, *JACM*, 41(1), 153, 1994.

[51] Demange, M., de Werra, D., Monnot, J., and Paschos, V. Th., Weighted node coloring: when stable sets are expensive, *Proc. 28th Int. Workshop on Graph Theoretical Concepts in Computer Science, WG'02*, Kučera, L., Ed., Lecture Notes in Computer Science, Vol. 2573, Springer, Berlin, 2002, p. 114.

[52] Demange, M., de Werra, D., Monnot, J., and Paschos, V. Th., Time slot scheduling of compatible jobs, Cahier du LAMSADE 182, LAMSADE, Universit Paris-Dauphine, 2001. Available on `http://www.lamsade.dauphine.fr/cahdoc.html#cahiers`.

[53] Demange, M., Monnot, J., and Paschos, V. Th., Maximizing the number of unused bins, *Found. Comput. Decision Sci.*, 26(2), 169, 2001.

[54] Bazgan, C., Monnot, J., Paschos, V. Th., and Serrire, F., On the differential approximation of MIN SET COVER, *Theor. Comput. Sci.*, 332, 497, 2005.

[55] Toulouse, S., Approximation Polynomiale: Optima Locaux Et Rapport Diffrentiel, Thse de doctorat, LAMSADE, Universit Paris-Dauphine, 2001.

[56] Demange, M., Monnot, J., and Paschos, V. Th., Differential approximation results for the Steiner tree problem, *Appl. Math. Lett.*, 733, 2003.

[57] Escoffier, B. and Paschos, V. Th., Differential approximation of MIN SAT, MAX SAT and related problems, *Eur. J. Oper. Res.*, to appear.

[58] Papadimitriou, C. H. and Yannakakis, M., Optimization, approximation and complexity classes, *JCSS*, 43, 425, 1991.

17
Hardness of Approximation

Mario Szegedy
Rutgers University

17.1 Introduction

This chapter is devoted to the core theory of inapproximability. Undoubtedly, the most fundamental part of the theory, with its numerous consequences, is the probabilistically checkable proofs (PCPs) theorem, which asserts that MAX-3SAT is NP-hard to approximate within a factor of $1 + \epsilon$ (for some $\epsilon > 0$). In Section 17.9 we sketch a recently obtained short proof to it [1].

Our survey places particular emphasis on the various kinds of reductions that are employed in the theory. We would like to convey our conviction that the entire theory is the study of these reductions and their compositions. We have found it important to introduce the reader to the proof and code-checking intuition. These play a key role in the theory, have been guiding its development, and even today, when alternative interpretations are available, still prove to be indispensable when trying to obtain stronger results.

Unfortunately, we had to make sacrifices to keep the size of the chapter within limits. We discuss only a handful of specific optimization problems. We could not have possibly opened the treasure chest of ad hoc inapproximability reductions, there are just so many of them. We have also omitted discussing how the syntax of optimization problems can often give a guideline to their (in)-approximability status. This subject, called the syntactic versus semantic view of (in)-approximability, is under heavy investigation and several advances have been reported recently [2,3]. We have found no room to convey recent excitement about the unique game conjecture and its consequences [4,5]. Finally, the chapter is concerned only with NP optimization problems. Probabilistic debate systems [6] and inapproximability of #P problems are out of the scope of this survey.

17.2 NP Optimization: Approximability and Inapproximability

Optimization problems are either *maximization* or *minimization* problems:

$$OPT(x) = \max_{y \in D(x)} F(x, y) \qquad \text{(maximization problem)}$$
$$OPT(x) = \min_{y \in D(x)} F(x, y) \qquad \text{(minimization problem)}$$

where $x \in \{0, 1\}^*$ is a string describing the input and $F(x, y)$ is a real-valued function (we often also assume nonnegativity). The *witness* y comes from a set that may depend on the input.

One may think of max and min as quantifiers. In analogy with NP, the class NPO is the set of those optimization problems for which $F(x, y)$ and the relation $y \in D(x)$ are polynomial-time computable. Here, the polynomial is in terms of $|x|$, the input length. We may get rid of the sometimes annoying $y \in D(x)$ domain condition by setting $F(x, y)$ definitely smaller (larger) than $OPT(x)$ if $y \notin D(x)$. This, however, might add extra complexity to the calculation of $F(x, y)$. NPO consists of NP maximization and NP minimization problems. To turn an NP maximization (minimization) problem into an NP problem we just augment the input with a threshold value and ask if OPT is larger (smaller) than the threshold.

While some important optimization problems are not in NPO, most of those that come from real life are. Examples are abundant: coloring, allocation, scheduling, Steiner tree problems, TSP, linear and quadratic programming, knapsack, vertex cover, etc. All of these examples (except linear programming) are NP-hard, and the best we can hope is to find quick approximate solution for them.

Approximation Ratio

Let $x \to A(x)$ $(\forall x : A(x) \in D(x))$ be a map. This map is said to *approximate* $OPT(x) = \max_{y \in D(x)} F(x, y)$ to within a factor of $r(x) \geq 1$ if

$$\forall x : OPT(x) \leq r(x) F(x, A(x))$$

The best such $r(x)$ is also called the *approximation ratio* achieved by A. If there is a polynomial-time computable A that achieves approximation ratio $r(x)$, we say that *OPT* is *approximable* within a factor of $r(x)$. When we seek to approximate OPT, we often choose $r(x)$ to be a function of the input length. If the input is a graph, $r(x)$ is typically chosen to be a function of the number of vertices or edges, but we could also make it dependent on the maximal degree, the girth, etc. When *OPT* is a minimization problem the bound in the above definition is replaced by $F(x, A(x)) \leq OPT(x) r(x)$. (In the literature sometimes $1/r(x)$ is called the approximation ratio. The two definitions can be told apart, since in our definition $r(x)$ is always greater than 1.)

Example 17.1 (Set cover)

Let x describe a polynomial size set system $\mathcal{S}$ (say, by listing the elements of each set in $\mathcal{S}$), let y describe a subsystem $\mathcal{S}' \subseteq \mathcal{S}$, and let $y \in D(x)$ iff $\cup \mathcal{S}' = \cup \mathcal{S}$. The set cover problem is asking to find $y \in D(x)$ such that $|\mathcal{S}'|$ is minimized. It can be shown that there is a polynomial-time algorithm that approximates the set cover problem within a factor of $1 + \ln |\cup \mathcal{S}|$.

Inapproximability

For the above example Feige has shown that set cover cannot be approximated within a factor of $(1 - \epsilon) \ln |\cup \mathcal{S}|$ in P for any fixed $\epsilon > 0$ unless $NP \subseteq DTIME(n^{\log \log n})$ [7]. In general, a statement that there is no polynomial-time algorithm for *OPT* with approximation ratio $r(x)$ under some complexity theoretic hypothesis is referred to as an *inapproximability result*, where $r(x)$ is called *inapproximability ratio*.

Feige's result is sharp in the sense that the set cover is approximable in polynomial time within a ratio of $C \ln |\cup \mathcal{S}|$ if and only if $C \geq 1$ (under our complexity theoretic hypothesis). Thus, $\ln |\cup \mathcal{S}|$ may be viewed as the approximation boundary of the set cover problem. In general, we call a function $r(x)$ an *approximation boundary* of problem OPT if OPT is polynomial-time approximable within a factor of $r(x)C$ for any $C > 1$, but OPT is (conditionally) hard to approximate within a factor of $r(x)C$ for any $C < 1$. The above is sometimes understood in the logarithmic sense, i.e., when $r(x)C$ is replaced by $r(x)^C$. The latter is clearly a weaker condition. For many NPO problems a type of *dichotomy* holds: approximating them beyond their approximation boundary is NP-hard. (The other alternative could be that the complexity of approximating them gradually increases as $r(x)$ decreases.)

For a long time no approximation boundaries were known for major NPO problems. The appearance of the theory of probabilistically checkable proofs (PCP theory) has changed this situation. In its rise, numerous exact inapproximability results (including the one above by Feige) were proven. This theory is the subject of our next sections.

17.2.1 The Emergence of the PCP Theory

Motivated by Graham's [8] exact bounds on the performance of various bin packing heuristics, Johnson [9] gave algorithms for the Subset Sum, the Set Cover, and the MAX k-SAT problems with guarantees on their performances ($1 + o(1)$, $O(\log |S|)$, $2^k/(2^k - 1)$, respectively). He also gave inapproximability results, but unfortunately they referred only to specific algorithms. Nevertheless, he has brought up the issue of classifying NPO problems by the best approximation ratio achievable for them in polynomial time. Although the goal was set, only a handful of inapproximability results existed. Sahni and Gonzalez [10] proved the inapproximability of the non-metric traveling salesman and some other problems (under $P \neq NP$). Garey and Johnson [11] introduced *gap amplification* techniques to show that the chromatic number of a graph cannot be approximated to within a factor of $2 - \epsilon$ unless $P = NP$, and an approximation algorithm for the max clique within *some* constant factor could be turned into an algorithm which approximates max clique within *any* constant factor.

The old landscape of approximation theory of NPO radically changed when in 1991 Feige et al. [12] for the first time used Babai et al.'s characterization of NEXP in terms of multiprover interactive proof systems [13] to show that approximating the clique within any constant factor is hard for NTIME($n^{1/\log\log n}$). Simultaneously, Papadimitriou and Yannakakis [14] defined a subclass of NPO, what they called MAXSNP, in which problems have an elegant logical description and can be approximated within a constant factor. They also showed that if MAX3SAT, vertex cover, MAX CUT, and some other problems in the class, could be approximated in polynomial time with an arbitrary precision, the same would hold for all problems in MAXSNP. They established this fact by reducing MAXSNP to these problems in an *approximation preserving* manner. They called their special reduction L-reduction and considered MAXSNP-completeness with respect to it a strong indication that a problem does not have *polynomial-time approximation scheme* (PTAS) (i.e., a sequence of polynomial-time algorithms achieving $1 + 1/k$ accuracy for $k = 1, 2, \ldots$). Their work showed great insight. What was missing was a relation between MAXSNP-completeness and usual hardness assumptions such as $P \neq NP$. In 1992, Arora et al. [15] showed that MAX3SAT is hard to approximate within a factor of $1 + \epsilon$ for some $\epsilon > 0$ unless $P = NP$. Their proof relied on PCPs, and employed several intricate arguments. They took techniques from Refs. [13,16–18], in particular, the important "proof recursion" idea of Arora and Safra [17]. The term PCP was also coined in the latter article. Rapid development came on the heals of these results:

1. Inapproximability of NPO problems.
2. Construction of approximation algorithms achieving optimal or near-optimal ratios (e.g., Ref. [19]).
3. A bloom of approximation preserving reductions and discovery of new (in)approximability classes.

PCP theory has turned out to be the key ingredient in determining the approximation boundaries of many NPO problems. Some problems remain open, like the Asymmetric Traveling Salesperson Problem, whose approximability status is not yet clarified. In a latest development, Dinur [1] gave a simplified proof for the ALMSS (Arora–Lund–Motwani–Sudan–Szegedy) theorem [20] (see a sketch in Section 17.9) eliminating much of the difficult algebra of the original proof.

17.3 Approximation-Preserving Reductions

Reduction is perhaps the most useful concept in algorithm design. Interestingly, it also turns out to be the most useful tool in proving computational hardness [21–23]. When in problem A Cook reduces to B, the hardness of B follows from the hardness of A. Unfortunately, Cook reduction does not ensure that if A is hard to approximate then B is hard to approximate. For reducing hardness of approximation new definitions are necessary.

Let $F_1(x, y)$ and $F_2(x', y')$ be functions that are to be optimized for y and y' (maximized or minimized in an arbitrary combination). Let $OPT_1(x)$ and $OPT_2(x')$ be the corresponding optimums. A Karp–Levin *reduction* involves two maps:

1. a polynomial-time map f to transform instances x of OPT_1 into instances $x' = f(x)$ of OPT_2 [Instance Transformation];
2. a polynomial-time map g to transform (input, witness) pairs (x', y') of OPT_2 into witnesses y of OPT_1.
   [Witness Transformation].

Observe that the witness transformation goes from OPT_2 to OPT_1. Let $opt_1 = OPT_1(x)$, $opt_2 = OPT_2(f(x))$, $appr_1 = F_1(x, g(f(x), y'))$, and $appr_2 = F_2(f(x), y')$.

The centerpiece of any *approximation-preserving* reduction scheme is a relation between these four quantities. This relation must express: "If $appr_2$ well approximates opt_2, then $appr_1$ well approximates opt_1." The first paper which defines an approximation preserving reduction was that of Orponen and Mannila [24]. Up to the present time more than eight notions of approximation preserving reductions exist differing only in the relation required between opt_1, opt_2, $appr_1$, and $appr_2$. For an example, consider the L-reduction of Papadimitriou and Yannakakis [14]. The required relations are $opt_2 \leq c_1 opt_1$ and $|appr_1 - opt_1| \leq c_2 |appr_2 - opt_2|$ for some constants c_1 and c_2. It easily follows from the next lemma, that L-reduction preserves PTAS.

Lemma 17.1

A reduction scheme preserves PTAS iff it enforces that

$$|appr_1 - opt_1|/opt_1 \rightarrow 0 \textit{ whenever } |appr_2 - opt_2|/opt_2 \rightarrow 0.$$

Proof

Here we only prove the "if" part. Assume we have a PTAS for OPT_2 and that OPT_1 reduces to OPT_2. To get a PTAS for $OPT_1(x)$ first we construct $f(x)$. Using the ϵ-approximation algorithm A_ϵ for OPT_2 we find a witness y' such that $(1 - \epsilon)OPT_2(f(x)) \leq F_2(f(x), y') \leq (1 + \epsilon)OPT_2(f(x))$. Hence $|F_2(f(x), y') - OPT_2(f(x))|/OPT_2(f(x)) \leq \epsilon$. When ϵ tends to 0, from the condition of the lemma we obtain that $|F_1(x, g(f(x), y')) - OPT_1(x)|/OPT_1(x)$ also tends to 0. Thus using f, g, and A_ϵ (for decreasing epsilon) we can build a sequence of algorithms that serves as a PTAS for OPT_1. □

The main advantage of approximation-preserving reductions is that they enable us to define large classes of optimization problems that behave in the same way with respect to approximation. A prominent example is MAXSNP: all problems in this class are constant-factor approximable, as shown via the L-reduction of Papadimitriou and Yannakakis.

17.4 Gap Problems, Karp Reductions, and the PCP Theorem

It soon became clear that besides approximation-preserving reductions, PCP theory also requires reductions between new types of problems, called *promise problems*. Promise problems are functions with three possible values: 0, 1, and "undefined." They occur as intermediate steps in reduction sequences from decision problems to functions. In PCP theory context Bellare et al. in Ref. [25] were the first to explain reductions through promise problems.

Let OPT be a minimization problem. Assume that for every input x we have two bounds: a lower bound $T_l(x)$ and an upper bound $T_u(x)$, both are polynomial-time computable in x. It is easy to see that if we can efficiently approximate $OPT(x)$ within a factor better than $r(x) = T_u(x)/T_l(x)$, then with only a polynomial (additive) overhead in the running time we can also solve:

- if $OPT(x) \geq T_u(x)$, the output is 0,
- if $OPT(x) \leq T_l(x)$, the output is 1,
- if $T_l(x) < OPT(x) < T_u(x)$, the output can be anything.

We call the above problem a *gap problem* and refer to it as $Gap(OPT, T_l, T_u)$. If OPT happens to be a maximization problem, the above definition stays valid, only the roles of 0 and 1 get exchanged.

Example 17.2

For graph G let $\chi(G)$ be the chromatic number of G. $OPT = \chi$ is a minimization problem. Let $T_l = 3$, $T_u = |V(G)|^{0.26}$, where $V(G)$ denotes the vertex set of the input graph. Karger, Motwani and Sudan solved $Gap(\chi, 3, |V(G)|^{0.26})$ in polynomial time. In fact, their algorithm well-colors any three chromatic graph using at most $|V(G)|^{0.26}$ colors. It is a famous open problem if for any $\epsilon > 0$ there is a polynomial-time algorithm that colors a three chromatic graph with $|V(G)|^{\epsilon}$ colors.

Witness giving condition: An algorithm for the problem $Gap(OPT, T_l, T_u)$, where OPT is an NP minimization problem, satisfies the Witness giving condition if for every x with $OPT(x) \leq T_l(x)$ the algorithm gives a "witness" y for which $F(x, y) < T_u(x)$. If OPT is a maximization problem then the witness giving condition requires that for every input x for which $OPT(x) \geq T_u(x)$, the algorithm computes a witness y for which $F(x, y) > T_l(x)$. Almost all polynomial-time solutions given to gap problems satisfy the witness giving condition.

Gap problems yield themselves to Karp reductions as explained below. For brevity we assume that OPT is an NP maximization problem.

Karp reduction from languages to gap problems

Let $L \subseteq \Sigma^*$ be a language. A Karp reduction from L to $Gap(OPT, T_l, T_u)$ is a polynomial-time map f from Σ^* to input instances of OPT such that

1. if $x \in L$, then $OPT(f(x)) \geq T_u(f(x))$;
2. if $x \notin L$, then $OPT(f(x)) \leq T_l(f(x))$.

For the above type of reduction the most prominent example is the PCP theorem:

Theorem 17.1 (PCP Theorem)

For some $\epsilon > 0$ it holds, that for every language $L \in NP$ there exists a polynomial-time computable function $f : \Sigma^* \to \{3CNF\ formulas\}$, such that

1. *if $x \in L$, then $f(x)$ is a formula in which all disjunctions are simultaneously satisfiable;*
2. *if $x \notin L$, then $f(x)$ is a formula in which one can satisfy at most $1 - \epsilon$ fraction of all clauses.*

Remark 17.1

The problem of maximizing the fraction of satisfied clauses of a 3CNF formula is called MAX-3SAT. *Formally, let $\phi = \bigwedge_{i=1}^{m}(y_{i,1}^{b_{i,1}} \vee y_{i,2}^{b_{i,2}} \vee y_{i,3}^{b_{i,3}})$ be a 3CNF formula. For an assignment y, define*

$$F(\phi, y) = |\{i \mid (y_{i,1}^{b_{i,1}} \vee y_{i,2}^{b_{i,2}} \vee y_{i,3}^{b_{i,3}})\textit{evaluates to true under assignment } y\}|/m.$$

Then $MAX\text{-}3SAT(\phi) = \max_y F(\phi, y)$.

We can restate Theorem 17.1 that any language in NP is Karp reducible to $Gap(MAX - 3SAT, 1 - \epsilon, 1)$.

Karp reduction between gap problems

Let OPT_1 and OPT_2 be maximization problems, T_l, T_u, T_l', T_u' polynomial-time computable functions. Let f be a polynomial-time computable function from the set of input instances for OPT_1 to the set of input instances for OPT_2 such that

1. if $OPT_1(x) \leq T_l(x)$, then $OPT(f(x)) \leq T_l'(f(x))$;
2. if $OPT_1(x) \geq T_u(x)$, then $OPT(f(x)) \geq T_u'(f(x))$.

Then f is a Karp reduction from $Gap(OPT_1, T_l, T_u)$ to $Gap(OPT_2, T_l', T_u')$. The above definition carries over without any difficulty to those cases when one or both of OPT_1 and OPT_2 are minimization problems. If such reduction exists and $Gap(OPT_1, T_l, T_u)$ is NP-hard then $Gap(OPT_2, T_l', T_u')$ is also NP-hard.

Example 17.3

For graphs G and H we denote by $G \times H$ their strong product: $V(G \times H) = V(G) \times V(H)$, $E(G \times H) = \{((u_1, v_1), (u_2, v_2)) \mid (u_1, u_2) \in E(\widehat{G}) \wedge (v_1, v_2) \in E(\widehat{H})\}$, where $\widehat{G}$ and $\widehat{H}$ are obtained from G and H by adding a loop to every node. Let $\omega(G)$ denote the maximum clique size of graph G. It is easy to see that $\omega(G \times H) = \omega(G)\omega(H)$. Let f be the map $G \to\ G \times G$ and let $l < u$ be arbitrary positive constants. The above implies that f is a Karp reduction from the gap problem $Gap(\omega, l, u)$ to $Gap(\omega, l^2, u^2)$.

The following scheme is a high-level description of the way we prove inapproximability results:

$$\begin{tikzcd}
\text{3SAT} \arrow[d, Rightarrow, "\text{PCP theorem (Karp reduction)}"] \\
\text{gap-3SAT} \arrow[d, Rightarrow, "\text{Karp reduction}"] \\
\text{Other gap problems} \arrow[d, Rightarrow, "\text{trivial}"] \\
\text{Corresponding inappr. results} \arrow[d, Rightarrow, "\text{Approx. preserving reduction}"] \\
\text{Inappr. results}
\end{tikzcd}$$

17.5 Probabilistic Verification: The FGLSS Graph

PCP theory grew out of the observation that probabilistic proof systems can be viewed as optimization problems. At the time of their discovery probabilistic proof systems were a surprising novelty. At the present time they represent a distinct contribution of the theory of computing to logic and mathematics. Their two origins are zero-knowledge proof systems [26]and Arthur–Merlin games [27].

Among all probabilistic proof systems, the so-called Multiprover interactive proof system (MIP) of Ben-Or et al. [28] is what we apply in the theory of PCPs. The polynomial-time randomized verifier of an MIP "interrogates" two (or more) noncommunicating all-powerful provers about the truth-hood of a statement. Instead of MIPs we consider a roughly equivalent system.

Probabilistic Oracle Machines (POM)

Let $M^y(x, r)$ be a probabilistic RAM with oracle y and random string r. M is said to accept a language L with completeness α and soundness β $(1 \geq \alpha > \beta \geq 0)$ iff

- if $x \in L$, then there is a y such that $Prob_r(M^y(x, r) = 1) \geq \alpha$;
- if $x \notin L$, then for every y it holds that $Prob_r(M^y(x, r) = 1) \leq \beta$.

The relevant parameters are the amount of randomness used ($|r|$), the query size (q), the completeness parameter (α), and the soundness parameter (β). The query size is the number of positions of y that M looks at for the worst-case input of size n. Every position of y contains an element of the Σ, the alphabet of y, which is assumed to be $\{0, 1\}$, unless otherwise said. If the alphabet size is larger, then $|\Sigma|$ is also a parameter. Recently, there is also an interest in minimizing the proof size, $|y|$ [1,29,30].

Probabilistically Checkable Proofs (PCP)

The witness y written on a POM machine's oracle tape is also called a *Probabilistically Checkable Proof.* The POM that checks the PCP is called a "PCP verifier." We use the terms "POM" and "PCP verifier" interchangeably. The latter term was born from an "anthropomorphic" interpretation of the verification process. We often speak in the first person when we describe the verifier's actions. Most PCP verifiers are nonadaptive: they ask all questions to the oracle at once.

To see the connection between proof systems and combinatorial optimization, let M be a POM with parameters $|r|$, q, α, β and consider the problem

$$OPT_M(x) = \max_y \text{Prob}_r(M^y(x, r))$$

Observe that if $x \in L$, then $OPT_M(x) \geq \alpha$ and if $x \notin L$, then $OPT_M(x) \leq \beta$, i.e., L Karp reduces to $Gap(OPT_M, \beta, \alpha)$. Thus, if L is NP-hard, approximating OPT_M within a factor better than α/β is also NP-hard. The significance of parameters $|r|$ and q will soon be clear.

Feige et al. have turned the $MIP = NEXP$ theorem of Babai, Fortnow, and Lund into an in-approximability result. It had to be scaled down first to the polynomial level, resulting in the statement: For an NP-complete language, L, there is a $C > 0$ and a probabilistic oracle machine M^* with query size randomness bounded by $(\log n)^C$, completeness parameter 1 and soundness parameter $1/n$. Hence OPT_{M^*} cannot be approximated by a factor of n. If this does not sound impressing, it is because computing OPT_{M^*} was not a frequently studied optimization problem. Feige et al. transformed OPT_{M^*} (or any OPT_M) into a maximum clique problem. Below we describe the transformation:

FGLSS (Feige–Goldwasser–Lovász–Safra–Szegedy) transformation

For a string $x \in \{0, 1\}^n$ and a probabilistic oracle machine $M^y(x, r)$ we define a graph $G_{x,M}$. The vertices of $G_{x,M}$ are ordered tuples (r, a) of 0-1 strings ($|a| = q$) such that M accepts if it has access to an oracle y that gives $a_1, a_2, \ldots, a_q$ for answers to the q subsequent queries of M, when we run it on inputs x and r. The main point is that given $a = a_1, a_2, \ldots, a_q$ we do not need the entire y to compute $M^y(x, r)$. Every oracle y that answers $a_1, a_2, \ldots, a_q$ to the k subsequent queries of $M^y(x, r)$ is said to be consistent with (r, a), as long as it also holds that $M^y(x, r)$ accepts. If $M^y(x, r)$ rejects, y is defined to be inconsistent with (r, a) for any a. Clearly, for fixed y and r if there is an a such that (r, a) is consistent with y, then this a is unique. For $r \neq r' \in \{0, 1\}^k$ and $a, a' \in \{0, 1\}^q$ we have an edge between (r, a) and (r', a') in $G_{x,M}$ if there is an oracle y consistent with both. The following is easy to see:

Lemma 17.2

Let $(r_1, a_1), (r_2, a_2), \ldots, (r_s, a_s)$ form a clique in $G_{x,M}$. Then there is an oracle y consistent with (r_i, a_i) for $1 \leq i \leq s$.

For a fixed oracle y the number of rs that are consistent with y (meaning that there exists an a such that (r, a) is consistent with y) is proportional with the probability over r that $M^y(x, r)$ accepts. Thus, $OPT_M(x)$ is proportional with the max clique size of $G_{x,M}$.

Since the number of all possible (r, a) pairs is $2^{|r|+q}$, graph $G_{x,M}$ has at most $2^{|r|+q}$ vertices. Applying this to the verifier M^* of Feige et al. we get that the number of vertices of G_{x,M^*} is at most $2^{2(\log n)^C}$. Since any algorithm that approximates OPT_{M^*} within a factor of n can solve NP, we can use any algorithm that approximates the max clique size of G_{x,M^*} within a factor of n to build an NP solver. The overhead is the cost of the FGLSS transformation, which is polynomial in the size of G_{x,M^*}. Expressing all parameters in terms of $N = |V(G_{x,M^*})|$, we get that for some constant C' the max clique problem of a graph with N nodes is not approximable in polynomial time within a factor of $2^{(\log N)^{1/C'}}$ unless $NP \subseteq DTIME(2^{(\log N)^{C'}})$. In [12,17,20,25,31–34] (starting with the original paper) the result was subsequently improved. The best current improvement is:

Theorem 17.2 (Zuckerman [34])

There are γ, $c > 0$ such that the maximal clique size of a graph with N nodes cannot be approximated within a factor of $\frac{N}{2^{(\log N)^{1-\gamma}}}$ in polynomial time unless NP $\subseteq DTIME(2^{(\log N)^c})$.

17.6 PCPs and Constraint Satisfaction Problems

We have seen that constructing a POM with small parameters has immediate inapproximability consequences. In Ref. [20] an even more dramatic consequence was found effecting the entire class MAXSNP:

Lemma 17.3

If for a language L there is a POM with perfect completeness, soundness $\gamma < 1$, logarithmic randomness and constant query size, then L Karp reduces to $Gap(MAX - 3SAT, \beta, 1)$ for some $\beta < 1$.

Indeed, let the POM in the lemma be $M^y(x, r)$. Since M queries at most $q = O(1)$ bits, for fixed x and r there is constant-size Boolean formula, $\Phi_{x,r}$ expressing if the verifier accepts or rejects the bits it views. We have

$$\max_y Prob(M^y(x, r) = 1) = \max_y |\{r|\Phi_{x,r}(y) = 1\}|/2^{|r|} \tag{17.1}$$

Constraint Satisfaction Problems (CSP)

The problem on the right-hand side of Eq. (17.1) is a MAX-CSP problem. A kCSP is a generalization of kSAT, where clauses can be any k-variable Boolean expressions. kCSPs are typically defined on Boolean variables, but it is easy to extend this definition to the case when the variables take values from a general constant-size alphabet Σ. In this case we talk about a $[k, \Sigma]$CSP.

Fact 17.1

Every PCP with completeness α, soundness γ, query size k, and witness-alphabet Σ can be turned into a $Gap(\mathrm{MAX} - [k, \Sigma]\mathrm{CSP}, \gamma, \alpha)$ instance.

In particular Eq. (17.1) reduces the L of Lemma 17.3 to $Gap(\mathrm{MAX} - [q, \Sigma]\mathrm{CSP}, \gamma, 1)$, where Σ is the alphabet of M. To prove Lemma 17.3 we need to reduce $Gap(\mathrm{MAX} - [q, \Sigma]\mathrm{CSP}, \gamma, 1)$ to $Gap(\mathrm{MAX} - 3\mathrm{SAT}, \gamma', 1)$. This is done by *gadgets* (in this case we have to transform little nondeterministic 3SAT formulas replacing the constraints of the k-CSP). Lemma 17.3 explains why we want to construct PCPs with constant number of check bits.

17.7 Codes and PCPs

Perhaps unexpectedly, when turning NP machines into POMs, it is not the machine, but rather the witness (or proof, in other words) that goes under a meaningful transformation. The old witness (let it be a coloring, a TSP tour, etc.) becomes a new and very interesting object, called PCP. (Of course, the machine needs to be adapted to the new checking task, but what motivates its actions is the presumed structure of the new witness.) To understand this better, let $\exists z \; N(x, z)$ be an NP machine, which we would like to transform into a POM that recognizes the same language. Without the loss of generality we can think of N as a machine for the chromatic number problem, x as a graph, and z as a coloring on the nodes. N verifies if z assigns different colors to all pairs of adjacent nodes in x.

Assume we manage to "encode" every witness (i.e., coloring) z into some string $y(x, z)$ (which may be viewed as a "probabilistically checkable version" of the same coloring) and design a probabilistic oracle machine $M^y(x, r)$ (also called checker or verifier) such that

1. if $N(x, z) = 1$, then $M^{y(x,z)}(x, r) = 1$ for every r.
2. for every x the string $y(x, z)$ (transformed from an original potential witness z) is an element of an error correcting code $\mathcal{C}_x$ that corrects δ fraction of errors.
3. if y is not δ-close to $\mathcal{C}_x$, then $Prob_r(M^y(x, r) = 1) \leq 0.5$.
4. if y is δ-close to some $y' \in \mathcal{C}_x$, but y' is not equal to some $y(x, z_0)$ for some z_0 for which $N(x, z_0) = 1$, then $Prob_r(M^y(x, r) = 1) \leq 0.5$.

Then M is a POM with completeness 1 and soundness 0.5 that recognizes the same language as N. The completeness property follows from 1. To see the soundness property assume that $\forall z \; N(x, z) = 0$. For some y how can $M^y(x, r)$ be accepting with probability greater than 0.5? From 3 we see that for this y has to be δ-close to the the code $\{y(x, z)|z\}$. But then there is a z_0 such that y is δ-close to z_0. Then 4 gives a contradiction, since $N(x, z_0) = 0$.

The above way of constructing PCPs gives a general philosophy. Point 3 calls for constructing codes such that a POM with small query size can tell with large certainty if a word is δ-far from a code word. In fact, the POM can be viewed as performing two procedures: (1) checking for closeness to the code and (2) given that y is close to the code, checking if it is an encoding of a witness that makes N accept. Sometimes these two tasks are merged together.

In Refs. [12,13,16] the technique to construct a PCP encoding code was *arithmetization.* When arithmetizing, the encoding function is the generalized Reed Solomon encoding. The PCP properties of this code are not straightforward. A technical detail, but important for the further developments, is that in these articles δ is not a constant, but rather inverse polylogarithmic.

In Ref. [17] the parameters were further reduced to $|q| = |r| = O(\log n)$ reaching an important milestone: NP was characterized for the first time as *PCP*$(\log n, \log n)$ (the two arguments are the number of the random bits and the number of the query bits).

In Ref. [20] the number of check bits had to be decreased to constant. Are there any error correcting codes over $\Sigma = \{0, 1\}$ that can be checked with constant number of queries even if we allow the code word to be exponentially long, and we do not require any additional properties? The answer is yes, and these codes play a fundamental role in the PCP theory.

Hadamard Code

Let $z \in \{0, 1\}^k$. We encode z as the sequence of all scalar products $had(z) = (z, v)_{v\in\{0,1\}^k}$. Here $(z, v) \overset{def}{=} \sum_i z_i v_i \bmod 2$. The above is known as the Hadamard encoding.

Lemma 17.4

Let $y = (y_v)_{v\in\{0,1\}^k}$ be any vector of length 2^k with elements from $\{0, 1\}$. Then

1. *If $y = had(z)$ for some $z \in \{0, 1\}^k$ then for every $v, t \in \{0, 1\}^k$ we have $y_v + y_t = y_{v+t}$.*
2. *If y is ϵ-far in Hamming distance from every word from the Hadamard code, then*

$$|\{(v, t)|y_v + y_t \neq y_{v+t}\}| \geq \epsilon\, 2^{2k}$$

The above lemma, which was first discovered with a slightly weaker constant by Blum et al. [18], gives a procedure to check membership in the Hadamard code: Pick $v, t \in \{0, 1\}^k$ randomly and independently, and reject if $y_v + y_t \neq y_{v+t}$. The lemma implies that if y is at least ϵ-far from all code words, the check gets rejected with probability at least ϵ. However, if it is a code word, the check gets accepted with probability 1. A further feature of the Hadamard code is that from any string y close to $had(z)$ we can recover any mod 2 subset-sum of the bits of z with high certainty, but we cannot easily recover nonlinear (in the two-element field arithmetic) functions of the encoded string. Arora et al. [20] employed the Hadamard encoding in their construction. The role of exponentially large codes in the PCP theory is made clearer in the next section.

We now describe another code, which is checkable with a constant number of bits and has the added nice feature that one can also recover any Boolean function of the encoded string with high certainty using only two queries. The checking procedure is remarkably efficient: three queries suffice. The code was invented by Bellare et al. [25] and was wonderfully analyzed by Hastad [35], who showed how this code plays a main role in obtaining sharp inapproximability results with it. It is also employed in the recent short proof for the PCP theorem.

The Long Code

Encode z ($z \in \{0, 1\}^k$) by the list of values that all Boolean functions take on z. We get *long*(z). Unlike the Hadamard code, the long code is nonlinear, although it is a subset of a Hadamard code. Since the number of Boolean functions that take k input bits is 2^{2^k}, the length of the long code is doubly exponential. The elements of the code are indexed by the names of the corresponding Boolean functions. There is another interpretation of the long code, which may be interesting. First encode z in unary and then encode $unary(z)$ with the Hadamard code. $long(z) = had(unary(z))$. If we have an abstract set S instead of $\{0, 1\}^k$ we may also encode its elements with the long code. Let $\mathcal{F}$ be the set of all functions $f : S \to \{0, 1\}$. For $z \in S$ we define $long_S(z) = (f(z))_{f\in\mathcal{F}}$, a string of length $2^{|S|}$.

Folding

If $y = long_S(z)$ then for every $f \in \mathcal{F}$ we have $y_{\neg f} = \neg y_f$. Let $s_0 \in S$ be an arbitrarily chosen element of S. For every f exactly one of $f(s_0)$, $\neg f(s_0)$ is 0. The folded long code $long'_S$ contains only those entries y_f of the long code for which $f(s_0) = 0$. Hence its words are from $2^{S\setminus\{s_0\}}$. When we describe a test for the folded long code it is often convenient to talk about y_f even when $f(s_0) \neq 0$. Under this, by definition, we mean $y_f = \neg y_{\neg f}$. To retrieve this value the tester needs to read only one bit from y. We call the above extension the *unfolding of y*. In PCP theory we always work with the folded code. Long code tests are studied by Fourier analytic techniques [31,35,36], and using the folded version gives nicer formulas. Also, if we unfold any string in $2^{S\setminus\{s_0\}}$ (not only a code word) then the unfolded string in 2^S will have Hamming weight exactly $2^{|S|-1}$.

Let $f_1 \vee f_2 \vee \cdots \vee f_l = 1$ and $y = long'(z)$ for some $z = \{0, 1\}^k$. Then $y_{f_1} \vee y_{f_2} \vee \cdots \vee y_{f_l} = 1$ must hold. This observation makes it possible to check $long'_s$ as follows:

Lemma 17.5 (Dinur [1])

For any finite set S the folded code $long'_S(z)$ can be checked looking at only three positions corresponding to functions whose OR is 1. *The first two functions of the triplet are randomly selected, while the third one is chosen by a random process (we do not give here the simple details) such that the OR of the three functions is* 1. *The procedure then accepts if $b_1 \vee b_2 \vee b_3 = 1$, where b_1, b_2, and b_3 are the three returned code bits. For the procedure the following holds:*

1. *If $y = long'_S(z)$, then y is accepted with probability* 1.
2. *If $dist(y, long'_S) > \delta$, then y is rejected with probability* 0.001δ.

17.8 Holographic Proofs and the Proof Recursion Idea

Babai et al. [16] have shown how to test randomly a (theorem, proof) pair without reading neither the theorem nor the proof. The theorem is assumed to be in an arbitrary error correcting format and the proof serves to check both the theorem and its format. When we say "proofs" and "theorems" normally we mean the corresponding notions of predicate calculus, but to understand this section one needs to think more generally (and in a sense, simpler). Ref. [16] considers *abstract proof systems*. In the most general case, the minimal requirement from a proof system is that it characterizes a set called *THEOREMS*, which comprises all "true statements." *THEOREMS* $\subseteq \{0, 1\}^*$ may be any language. The notion is merely syntactic. A slightly less minimal requirement is that *THEOREMS* be recursively enumerable. If so, then there is a polynomial-time machine N such that $x \in$ *THEOREMS* if and only if $\exists x\ N(x, y)$. Here, y can be arbitrarily long compared with x. We prefer to rename x and y and write the above as $\exists$ proof N(theorem, proof). We would like to emphasize that theorems and proofs are only strings. No semantics is attached to them. The only way to get convinced if a (theorem, proof) pair is valid if we run N on it.

The next step in Ref. [16] is to introduce the notion of "holographic" proof systems. Let us assume that the theorems of a conventional (nonholographic) abstract proof system form the set *THEOREMS*$_0$. Let $E_k : \{0, 1\}^k \to \{0, 1\}^{h(k)}$ be a family of efficiently computable error correcting encoding that correct δ fraction of errors. Let $E(x) = E_{|x|}(x)$. We define a new set

$$THEOREMS = \{E(x)|x \in THEOREMS_0\}$$

This transformation of the theorem set makes it error-resistant: Even if we delete or change a constant ($< \delta$) fraction of a theorem $\in$ *THEOREMS*, we can recover it using error correction. More importantly, if $THEOREMS_0$ is recursively enumerable (RE), then Ref. [16] shows how to build a "holographic proof system" for *THEOREMS* (below L stands for $THEOREMS_0$):

Theorem 17.3 (Babai et al. [16])

Let $E_k : \{0, 1\}^k \to \{0, 1\}^{h(k)}$ be a family of efficiently computable error correcting encodings that correct δ fraction of errors. For every $L \in RE$ with verifier N, there is a probabilistic oracle machine $M^{z,y}(k, r)$ (k is the length of the unencoded theorem, $z \in \{0, 1\}^{h(k)}$) such that

1. *If $x \in L$, $|x| = k$, then any witness y_0 for which $N(x, y_0) = 1$ can be turned into a witness y for M for which $M^{E(x),y}(k, r) = 1$ for every r. The transformation takes polynomial time in $|y_0|$.*
2. *Let $THEOREMS_k = \{E(x)|x \in L \cap \{0, 1\}^k\}$. For every z with $dist(z, THEOREMS_k) > \delta\, h(k)$ and for every y we have $Prob_r(M^{z,y}(k, r) = 1)) \leq 1/2$.*
3. *The query size and $|r|$ are only polylogarithmic in $|z| + |y|$.*

If we allow exponential blow-up in the parameters, then a much simpler holographic proof checker was found by Dinur [1], which she uses in her simple proof for the PCP theorem:

Assignment Tester of Dinur

Let F be an arbitrary Boolean function on k variables. Let $E : \{0, 1\}^k \to \{0, 1\}^l$ be an arbitrary encoding that corrects 10% of errors, and let

$$THEOREMS \overset{def}{=} \{E(x)|x \in \{0, 1\}^k;\quad F(x) = 1\}$$

The Dinur assignment tester is a holographic proof system for $z \in$ *THEOREMS*. Dinur's holographic proof for some $E(x) \in THEOREMS$ is simply $y = long'_{THEOREMS}(E(x))$. Given a candidate (z, y) for a (theorem, proof) pair, the verifier checks that the following hold:

1. y is close to some member of $long'_{THEOREMS}$;
2. assuming that the first condition holds, y is close to $long'_{THEOREMS}(z)$.

The second check is the reason why z of the true prover needs to come from an error correcting code. The first check is done as in Lemma 17.5. The second check is done by selecting a random index $1 \leq \rho \leq l$ and checking if the ρth bit of z is correctly encoded in y. For this the verifier picks a random $f : \{0, 1\}^l \to \{0, 1\}$ and checks if $y_f + y_{f+\pi_\rho} = z_\rho$, where π_ρ is the function that projects a word to its ρth bit: $w \to w_\rho$. Remark: π_ρ behaves as any other Boolean valued function on *THEOREMS*. In particular, the folded long code lets it to recover via the expression $y_f + y_{f+\pi_\rho}$, where f is arbitrary.

The verifier, instead of checking both 1 and 2, selects either 1 or 2 with 50% probability. If the above verification process is accepting with probability 0.999, then y is close to some $long'_{THEOREMS}(z')$. Also, z' is close to z in Hamming distance. This in turn means that z decodes to $x = E^{-1}(z')$, where $F(x) = 1$.

Reducing the Query Size

The query size of the above procedure is 3. As we see in the next section, it has a definite advantage to reduce the number of queries of the verifier to two even at the price of increasing the alphabet size to eight. To this end Dinur employs the following nice standard trick. To y we add another proof, y', that contains all the triplets of bits that are potentially read by the checking procedure. The alphabet of y' is $\{0, 1\}^3$. The verifier proceeds according to the same protocol as before, but when a check is to be performed, it reads the corresponding triplet, τ, from y'. To make sure that τ faithfully describes the corresponding three bits of $z \cup y$, it randomly selects one of the three bits of τ, compares it with the corresponding bit in $z \cup y$, and rejects if τ lies about it. The following is not hard to show:

Lemma 17.6

There is fixed $c > 0$ such that if z is δ-far from THEOREMS then for every y, y' the above 2-query verifier rejects $(z, y \cup y')$ with probability at least $c\delta$. On the other hand, if $z \in THEOREMS$ then there exist y, y' such that the verifier accepts $(z, y \cup y')$ with probability 1.

The holographic proof idea was slightly generalized in Refs. [1,10,37,38].

Proof Recursion

Arora and Safra [17] were the first to describe *proof recursion*, a method in PCP building to decrease the query size. Proof recursion builds on the holographic proof idea. Below we explain it in its simplest form. In this form proof recursion is used to decrease the size of the alphabet of the witness tape rather than the query size. In the process the query size should not increase by much. If the alphabet size is small but the query size is large, to do the proof recursion, we first need another transformation that trades query size for alphabet size by reducing the former and increasing the latter (see a simple example above). In general, if the query size is nonconstant, to achieve this trade-off may be quite technical.

Outer Verifier

Let M be a POM, called *outer verifier*, that makes a constant number of queries, but its witness is over a large alphabet, Σ_{big}. Let E be a binary error correcting encoding of Σ_{big} that corrects constant fraction of errors. From a witness $(y_i)_{1\leq i\leq m}$ we create another witness $(E(y_i))_{1\leq i\leq m}$, i.e., we encode every symbol of the witness (or proof) individually and concatenate the resulting code words.

Inner Verifier

To check the new witness we use a verifier that works under the holographic proof principles. Let us assume that the outer verifier for a random string r would read $y_{i_1}, y_{i_2}, \ldots, y_{i_q}$ and accept if some Boolean predicate $F(y_{i_1}, \ldots, y_{i_q})$ equals 1. Define:

$$THEOREMS = \{E(\sigma_1)E(\sigma_2)\ldots E(\sigma_q)|F(\sigma_1, \sigma_2, \ldots, \sigma_q) = 1\}$$

Note that if E corrects δ fraction of errors, then $E^q = E \times \cdots \times E$ still corrects δ/q fraction of errors, so *THEOREMS* is in an error correcting format (recall that by our assumption q is a constant). Therefore, a holographic proof checker capable of recognizing *THEOREMS* can be applied. (BFLS, D, and other constructions ensure the existence of such proof checkers for a wide range of parameters.) This proof checker, also called the *inner verifier*, needs to have access to a holographic proof besides $E(y_1)E(y_2)\ldots E(y_q)$.

Composed Verifier

We need to build an inner verifier for every fixed r as follows. The *composed verifier*, given r, first computes the indices of the check bits of the outer verifier, and the Boolean predicate F_r that represents the acceptance condition of the outer verifier for this r. From F_r it determines $THEOREMS_r$ and computes a random query of the associated inner verifier. For this the composed verifier uses an additional random string r' with length that agrees with the randomness required by the inner verifier. Then the composed verifier makes the queries. This is the only occasion when it reads anything from the witness tape. Thus the query and alphabet sizes of the composed verifier are that of the inner verifier. The composed proof itself is $(E(y_i))_{1\leq i\leq m} \cup \bigcup_r \text{holo}_r$, where holo_r is the holographic proof associated with random string r.

Building an outer verifier is not trivial, and it is a main technical component in Ref. [17]. Arora et al. [20] build a verifier that is both inner and outer, and which needs only logarithmic randomness and polylogarithmic query bits. They compose this verifier with itself a constant number of times. The verifier has a natural size parameter, which is set differently throughout the levels of composition. In particular, the size rapidly shrinks in every new iteration. For the last time they use a different inner verifier whose alphabet size is 2, and query size is constant similarly to Dinur's assignment tester. The latter was a novel idea at the time. To achieve significant improvement of the parameters in just a constant number of iterations the outer verifier must be very powerful, which makes the ALMSS [20] construction involved. To keep the number of iterations constant appeared to be necessary, because in their case every iteration of the proof recursion either added a constant to the query size or shrunk the completeness-soundness gap. In Refs. [1,38] gap-increasing reductions help to get around this. Shortly after Ref. [20] a powerful gap-increasing reduction, called *parallel repetition* was invented by Raz [39], and exploited in constructions to obtain sharp inapproximability results [35]. The parallel repetition, unlike the reductions in [1,38], which only add a constant to $|r|$, increases $|r|$ by a constant factor, so it can be used only (essentially) once without making $|r|$ superlogarithmic.

17.9 A Short Proof of the PCP Theorem

Before Refs. [1,38] several different proofs have been made for the PCP theorem, but their rough structure did not differ much. One of the nice features of the new proof is that it can be almost fully explained without mentioning PCPs, and talking only about Gap-CSPs (see Section 17.6). In particular, Dinur proves the following form of the PCP theorem:

Theorem 17.4

Let $\Sigma = \{0, 1\}^3$. *Then* $[2, \Sigma]$*CSP Karp reduces to Gap*$(MAX - [2, \Sigma]CSP, 0.9999, 1)$ *in polynomial time. (Note:* $[2, \Sigma]$*CSP is NP-complete.)*

For the rest of the section we fix $\Sigma = \{0, 1\}^3$.

Instance Size and Satisfiability Gap

For a CSP $\Phi = \bigwedge_{i=1}^{m} \Phi_i$, define the *instance size* as $|\Phi| = m$. The *satisfiability gap* of Φ is $\overline{\text{sat}}(\Phi) = \min_y |\{i|\Phi_i(y) = 0\}|/m$ (one minus the maximal fraction of the simultaneously satisfiable constraints). In words, Eq. (17.1) says

Let Φ be a $[k, \Sigma]$CSP instance. Either $\overline{\text{sat}}(\Phi) = 0$ (the formula is satisfiable) or $\overline{\text{sat}}(\Phi) \geq 1/|\Phi|$ (the formula is not satisfiable). The satisfiability gap cannot be smaller than $1/|\Phi|$ since if the formula is not satisfiable then at least one component of Φ is not satisfied under any assignment. Dinur constructs a reduction that enlarges this tiny gap to a constant, while keeping the gap of satisfiable instances 0. The reduction makes small progresses at a time.

It is sufficient to show that any $[2, \Sigma]$CSP instance Φ can be reduced in polynomial time to a $[2, \Sigma]$CSP instance Φ' with the following properties:

1. if Φ is satisfiable then Φ' is satisfiable.
2. $|\Phi'| \leq 10^{10^{10^{googol}}} |\Phi|$
3. $\overline{\text{sat}}(\Phi') \geq \min\{2\ \overline{\text{sat}}(\Phi),\ 0.0001\}$.

We say that a $[2, \Sigma]$CSP instance is d-regular, expanding, if the graph we obtain by replacing each constraint with the corresponding pair of variables is a d-regular expander. Let Φ be an arbitrary $[2, \Sigma]$CSP. Let $d = 11$ and t be a constant to be determined later. We obtain Φ' from Φ in three steps:

$$\begin{array}{lcl}
\Phi \in [2, \Sigma]\text{CSP} & \rightarrow & \text{(to improve on the structure)} \\
\Phi_{reg} \in d\text{-regular, expanding } [2, \Sigma]\text{CSP} & \rightarrow & \text{(to gain the gap)} \\
\Phi_{big} \in [2, \Sigma^{(d+1)^{\lceil \frac{t}{2} \rceil}}]\text{CSP} & \rightarrow & \text{(to reduce the alphabet size)} \\
\Phi' \in [2, \Sigma]\text{CSP} & &
\end{array}$$

Furthermore: (1) If Φ is satisfiable then so are Φ_{reg}, Φ_{big}, and Φ', (2) Each reduction increases the size of the instance only by a constant factor: $|\Phi_{reg}| \leq 30|\Phi|$, $|\Phi_{big}| = (d+1)^{\lceil \frac{t}{2} \rceil - 1}|\Phi_{reg}|$, $|\Phi'| \leq 2^{2^{31(d+1)^{\lceil \frac{t}{2} \rceil}}} |\Phi_{big}|$, and (3) The satisfiability gaps change as

$$\begin{aligned}
\overline{\text{sat}}(\Phi_{reg}) &\geq 0.1\ \overline{\text{sat}}(\Phi) \\
\overline{\text{sat}}(\Phi_{big}) &\geq \min\{200000\ \overline{\text{sat}}(\Phi_{reg}),\ 0.001\} \\
\overline{\text{sat}}(\Phi') &\geq 0.0001\ \overline{\text{sat}}(\Phi_{big})
\end{aligned}$$

Only for the $\Phi_{big} \rightarrow \Phi'$ reduction we need PCP ideas, although very little.

Reducing the Alphabet Size

In fact, Sections 17.7 and 17.8 have already prepared us to understand this reduction. Since $\Sigma = \{0, 1\}^3$, we can identify $\Sigma_{big} \overset{def}{=} \Sigma^{(d+1)^{\lceil \frac{t}{2} \rceil}}$ with $\{0, 1\}^{3(d+1)^{\lceil \frac{t}{2} \rceil}}$. If Φ_{big} is a positive instance (i.e., satisfiable), we encode each letter of the satisfying assignment with some encoding function $E : \{0, 1\}^{3(d+1)^{\lceil \frac{t}{2} \rceil}} \rightarrow \{0, 1\}^{30(d+1)^{\lceil \frac{t}{2} \rceil}}$ that corrects constant fraction of errors (from coding theory we know that such encoding exists). If v_i and v_j are two variables on which Φ_{big} has a constraint, we install a Dinur Assignment Tester (see the previous section, Lemma 17.6) which checks the property:

$$THEOREMS_{i,j} = \{(E(\sigma), E(\sigma'))|(\sigma, \sigma') \in \Sigma^2_{big} \text{ satisfy all constraints on } v_i, v_j\}$$

The proof, $(E(\sigma), E(\sigma'))$, is encoded, as required. A little detail is that the encoding is $E \times E$, rather than E, but $E \times E$ still corrects constant fraction of errors. Let us denote the holographic proof of this assignment tester by $\text{holo}_{i,j}$, and the set of all i, j pairs for which such a tester is built, by H. The properties of the Dinur Assignment Tester guarantee that when the constraint is satisfied, and the prover is faithful to the protocol, all tests are accepting. The tester looks at binary constraints over the alphabet $\Sigma = \{0, 1\}^3$, therefore, by Fact 17.1 it can be turned into a $[2, \{0, 1\}^3]$*CSP*. This CSP is made just for a single constraint of Φ_{big}. To obtain Φ' we unite the above sets for all constraints of Φ_{big}. The variables are all bits of all encoded variables of Φ_{big} together with $\bigcup_{i,j\in H} \text{holo}_{i,j}$.

Assume now that Φ_{big} is a negative instance. It takes a little argument to show that in this case the satisfiability gap of Φ' constructed above is at least $0.0001\overline{\text{sat}}(\Phi)_{big}$, *independently of t*. The transformation blows up the instance size by only a constant factor (dependent of t, but it is all right). These were all we wanted to achieve in this stage.

Making It Regular, Expanding

The $\Phi \rightarrow \Phi_{reg}$ reduction is even simpler. It is a fairly standard transformation which involves creating deg v clones of every node v of the constraint graph of Φ. We distribute the outgoing edges among the clones: Each clone remains connected to exactly one outgoing edge. For every v we fit a degree d' expander (say, with $d' = 5$) on its clones putting equality constraints on the new edges. This way we obtain a $d' + 1$-regular graph, which may not be an expander itself (recall that we did not have any assumption on the original graph). To ensure the expanding property, we now add new edges with empty constrains on them. The new edges form a degree d' expander on the entire vertex set. The final graph is $d = 2d' + 1$-regular and it is an expander. Throughout the whole construction we preserve multiple edges (possibly with different constraints).

Powering

The second reduction is a wonderful new addition to *PCP* theory and this is the one that really gains us the gap. We define an operation on binary constraint systems, called *powering*. Let G be a constraint graph and $t > 1$ be an integer. First we add a loop to each node (with an empty constraint). We denote the resulting graph with $G + I$. Then we construct $(G + I)^t$ in such a way that

- The vertices of $(G + I)^t$ are the same as the vertices of G.
- Edges: u and v are connected by k edges in $(G + I)^t$ iff the number of t steps paths from u to v in $G + I$ is exactly k.
- Alphabet: The alphabet of $(G + I)^t$ is $\Sigma^{(d+1)^{\lceil t/2 \rceil}}$, where every vertex specifies values for all of its neighbor reachable in $\lceil t/2 \rceil$ steps.
- Constraints: The constraint associated with an edge (u, v) of $(G+I)^t$ is satisfied iff the assignments for u and v are consistent with an assignment that satisfies all of the constraints induced by the $\lceil t/2 \rceil$ neighborhoods of u and v.

If G is satisfiable then $(G + I)^t$ is satisfiable as well. More importantly, powering has the following satisfiability gap enlarging property:

Lemma 17.7 (**Amplification Lemma** [1])

Let Σ be an arbitrary constant-size alphabet. There exists a constant $\gamma = \gamma(d, |\Sigma|) > 0$ such that for any $t > 0$ and for any d-regular expanding constraint graph G:

$$\overline{\text{sat}}((G + I)^t) \geq \gamma\sqrt{t} \min\left\{\overline{\text{sat}}(G), \frac{1}{t}\right\}$$

We define $\Phi_{big} = (\Phi_{reg} + I)^t$. Parameter t has to be chosen so that we get a large enough $\overline{\text{sat}}(\Phi_{big})/\overline{\text{sat}}(\Phi_{reg})$ ratio to compensate for the loss in the satisfiability gap in the first and third transformations, and even gaining a factor of 2 over that. This can be ensured because of Lemma 17.7.

Many of the stated properties of the reductions are intuitively clear, although some require nontrivial arguments. To get Theorem 17.4 we need to cycle through the reduction set $-log_2\alpha$ times (or, if we do not know α, more), where α is the satisfiability gap of the very first instance. $-log_2\alpha \leq \log m$, so the size of the final instance is polynomially bounded by that of the first, where the size of the latter is m.

References

[1] Dinur, I., The PCP theorem by gap amplification, Electronic Colloquium on Computational Complexity, Technical Reports, 2005.

[2] Khanna, S., Motwani, R., Sudan, M., and Vazirani, U., On syntactic versus computational views of approximability, *SIAM J. Comput.*, 28, 164, 1998.

[3] Khanna, S., Sudan, M., Trevisan, L., and Williamson, D. P., The approximability of constraint satisfaction problems, *SIAM J. Comput.*, 30(6), 1863, 2000.

[4] Khot, S., On the power of unique 2-prover 1-round games, *Proc. of STOC*, 2002, p. 767.

[5] Trevisan, L., Approximation algorithms for unique games, Electronic Colloquium on Computational Complexity, Technical Reports, 2005.

[6] Condon, A., Feigenbaum, J., Lund, C., and Shor, P., Random debaters and the hardness of approximating stochastic functions, *SIAM J. Comput.*, 26(2), 369, 1997.

[7] Feige, U., A threshold of ln n for approximating set cover, *JACM*, 45(4), 634, 1998. (Prelim. version in STOC'96.)

[8] Graham, R. L., Bounds for certain multiprocessing anomalies and related packing algorithms, *Proc. of the Spring Joint Conf.*, 1972, p. 205.

[9] Johnson, D., Approximation algorithms for combinatorial problems, *JCSS*, 9, 256, 1974.

[10] Sahni, S. and Gonzalez, T. F., P-complete approximation problems, *JACM*, 23(3), 555, 1976.

[11] Garey, M. and Johnson, D., The complexity of near-optimal graph coloring, *JACM*, 23, 43, 1976.

[12] Feige, U., Goldwasser, S., Lovász, L., Safra, S., and Szegedy, M., Interactive proofs and the hardness of approximating cliques, *JACM*, 43(2), 268, 1996.

[13] Babai, L., Fortnow, L., and Lund, C., Non-deterministic exponential time has two-prover interactive protocols, *Comput. Complexity*, 1, 3, 1991.

[14] Papadimitriou, C. H. and Yannakakis, M., Optimization, approximation, and complexity classes, *JCSS*, 43, 425, 1991.

[15] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M., Proof verification and hardness of approximation problems, *Proc. of FOCS*, 1992.

[16] Babai, L., Fortnow, L., Levin, L. A., and Szegedy, M., Checking computations in polylogarithmic time, *Proc. of STOC*, 1991.

[17] Arora, S. and Safra, S., Probabilistic checking of proofs: a new characterization of NP, *JACM*, 45(1), 70, 1998. (Prelim. version in FOCS'92.)

[18] Blum, M., Luby, M., and Rubinfeld, R., Self-testing/correcting programs with applications to numerical problems, *JCSS*, 47(3), 549, 1993.

[19] Goemans, M. and Williamson, D., Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *JACM*, 42(6), 1115, 1995.

[20] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M., Proof verification and the hardness of approximation problems, *JACM*, 45, 1998. (Prelim. version in FOCS'92.)

[21] Cook, S., The complexity of theorem-proving procedures, *Proc. of STOC*, 1971.

[22] Karp, R. M., Reducibility among combinatorial problems, in *Complexity of Computer Computations*, Plenum Press, Miller, R. E. and Thatcher, J. W., Eds. New York, 1972, p. 85.

[23] Levin, L., Universal'nyĭe perebornyĭe zadachi (universal search problems: in Russian), *Problemy Peredachi Informatsii*, 9(3), 265, 1973. A corrected English translation appears in an appendix to Trakhtenbrot.

[24] Orponen, P. and Mannila, H., On Approximation Preserving Reductions: Complete Problems and Robust Measures, TR C-1987-28, Department of Computer Science, University of Helsinki, 1987.

[25] Bellare, M., Goldreich, O., and Sudan, M., Free bits, PCPs, and nonapproximability—towards tight results, *SIAM J. Comput.*, 27(3), 804, 1998.

[26] Goldwasser, S., Micali, S., and Rackoff, C., The knowledge complexity of interactive proof systems, *SIAM J. Comput.*, 18(1), 186, 1989.

[27] Babai, L. and Moran, S., Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes, *JCSS*, 36, 254, 1988.

[28] Ben-Or, M., Goldwasser, S., Kilian, J., and Wigderson, A., Multi-prover interactive proofs: how to remove intractability, *Proc. of STOC*, 1988, p. 113.

[29] Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., and Vadhan, S. P., Robust PCPS of proximity, shorter PCPS and applications to coding, *Proc. of STOC*, 2004, p. 1.

[30] Ben-Sasson, E. and Sudan, M., Simple PCPS with poly-log rate and query complexity, *Proc. of STOC*, 2005, p. 266.

[31] Hastad, J., Clique is hard to approximate within $n^{1-\epsilon}$, *Acta Mathematica,* 182, 105, 1999. (Prelim version in FOCS'96.)

[32] Engebretsen, L. and Holmerin, J., Towards optimal lower bounds for clique and chromatic number, *Theor. Comput. Sci.*, 299, 2003.

[33] Khot, S., Improved inaproximability results for maxclique, chromatic number and approximate graph coloring, *Proc. of FOCS*, 2001, p. 600.

[34] Zuckerman, D., Linear degree extractors and the inapproximability of max clique and chromatic number, *Proc. of STOC*, 2006.

[35] Hastad, J., Some optimal inapproximability results, *JACM*, 48(4), 798, 2001. (Prelim. version in STOC'97.)

[36] Hastad, J., Testing of the long code and hardness for clique, *Proc. of STOC*, 1996.

[37] Szegedy, M., Many-valued logics and holographic proofs, *Proc. of ICALP*, 1999.

[38] Dinur, I. and Reingold, O., Assignment testers: towards a combinatorial proof of the PCP-theorem, *Proc. of FOCS*, 2004, p. 155.

[39] Raz, R., A parallel repetition theorem, *SIAM J. Comput.*, 27(3), 763, 1998.

II

Local Search, Neural Networks, and Metaheuristics

18
Local Search

Roberto Solis-Oba
The University of Western Ontario

18.1 Introduction

Typically, a combinatorial optimization problem involves a set E of elements (called *ground set*) and the goal is to arrange, group, order, or select a subset of elements from E that optimizes a given objective function. Classical examples of combinatorial optimization problems include the minimum spanning tree problem, the shortest paths problem, and the traveling salesman problem [1].

Local search is perhaps one of the most natural ways to attempt to solve a combinatorial optimization problem. The idea of local search is simple: Given a (probably not very good) solution s for a combinatorial optimization problem, try to improve the value of the solution by making "local changes" to s. A local change might involve, for example, adding elements from the ground set to s, removing elements from s, changing the way in which elements are grouped in s, or changing the order of the elements in s. If an improvement can be achieved in this manner, then a new solution s' is obtained. This process is continued until no further improvement can be obtained.

Local search has been successfully used to find good solutions for a large number of complex problems, the most famous of them probably being the traveling salesman problem [1]. The empirical performance of local search algorithms has been extensively studied for a large number of problems in scheduling, Very Large Scale Integration design, network design, distributed planning and production control, and many other fields [2]. Most of these studies concluded that local search is a good method for efficiently computing near-optimum solutions to problems of realistic sizes (see, e.g., Refs. [2,3]). In this chapter we explore the use of local search in the design of approximation algorithms with provable performance guarantee for NP-hard combinatorial optimization problems.

The idea of local search might be better understood by considering an example. In the *multiprocessor scheduling problem* the goal is to schedule a set $J = \{j_1, j_2, \dots, j_n\}$ of jobs into a group $M = \{M_1, M_2, \dots, M_m\}$ of m identical machines so that the completion time of the last job, also called the *makespan* of the schedule, is minimized. Each job j_i has a processing time p_i, and every machine can process only one job at a time. Furthermore, we assume that the processing of a job cannot be interrupted (i.e., preemptions are not allowed).

Let us consider a specific instance of the multiprocessor scheduling problem. Let $J = \{j_1, j_2, j_3, j_4, j_5, j_6\}$ with processing times $p_1 = 3$, $p_2 = 2$, $p_3 = 3$, $p_4 = 4$, $p_5 = 2$, and $p_6 = 1$, and $M = \{M_1, M_2, M_3\}$.

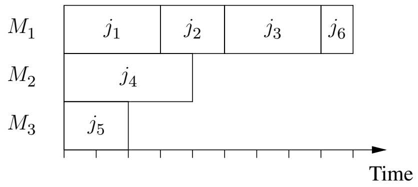


FIGURE 18.1 A schedule for J.

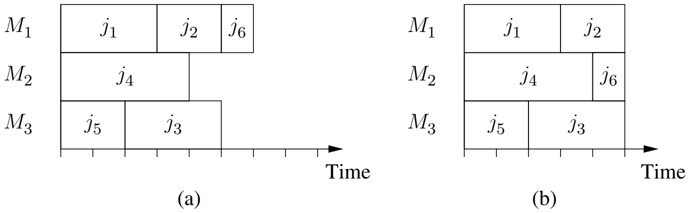


FIGURE 18.2 Local improvements for the solution in Figure 18.1.

A solution for this instance of the multiprocessor scheduling problem is shown in Figure 18.1. Machine M_1 processes jobs j_1, j_2, j_3, and j_6 requiring total processing time $3 + 2 + 3 + 1 = 9$. This is the *load* of machine M_1. The loads of M_2 and M_3 are 4 and 2, respectively. The makespan or length of this schedule is equal to the completion time of the last job, and it is also equal to the maximum machine load, namely 9.

We can make local changes to this solution to try to improve the makespan. For example, we could move a job from one machine with maximum load to a machine with minimum load. If, say, we move j_3 from M_1 to M_3, we get the solution shown in Figure 18.2(a) with makespan 8. As this solution is better than the first one, we keep it and try to further improve it. Now we can move j_6 to M_3 to get the solution depicted in Figure 18.2(b) with makespan 5.

This last solution cannot be improved by moving any one of the jobs to a different machine. In fact, one can show that this is an optimum solution for the problem since the total processing time of all the jobs is 15 and, thus, 3 machines need at least $15/3 = 5$ units of time to process them all.

18.1.1 Local Search and Combinatorial Optimization

Formally, a combinatorial optimization problem Π consists of a collection of instances $(\mathcal{S}, c)$. For each instance $(\mathcal{S}, c)$, $\mathcal{S}$ is the set of *feasible solutions*, and it consists of a family of subsets from a finite ground set E. The second component c of an instance is an objective function $c : \mathcal{S} \to I\!R$. The goal of a combinatorial optimization problem is to find a solution $s^* \in \mathcal{S}$ with minimum or maximum objective value, i.e.,

$$c(s^*) = \underset{s \in \mathcal{S}}{\texttt{optimum}}\, c(s)$$

where `optimum` is either `min` or `max`.

A *neighborhood function* $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$ specifies for each solution $s \in \mathcal{S}$ a subset $\mathcal{N}(s)$ of neighbors of s, or solutions that are "close" to s. The local search algorithm that we informally described above is called the *iterative improvement* algorithm.

Algorithm IterativeImprovement ($\mathcal{S}$, $\mathcal{N}$, c)

In: Set $\mathcal{S}$ of feasible solutions, neighborhood function $\mathcal{N}$, and objective function c.
Out: A local optimum solution $s \in \mathcal{S}$ with respect to $\mathcal{N}$ and c.

```
1. Compute an initial feasible solution $s \in \mathcal{S}$.
2. while $\mathcal{N}(s)$ contains a better solution than $s$ do {
```

```
3.        Choose a solution $s' \in \mathcal{N}(s)$ with better value $c(s')$ than $c(s)$.
4.        Set $s \leftarrow s'$
  }
5. Output $s$.
```

The solution s computed by the algorithm has the best possible value among all the solutions in its neighborhood $\mathcal{N}(s)$

$$c(s) = \underset{s' \in \mathcal{N}(s)}{\texttt{optimum}}\, c(s') \tag{18.1}$$

Therefore, this solution s is called a *local optimum* with respect to $\mathcal{N}$ and c. The set of local optimum solutions in the feasible solution set $\mathcal{S}$ with respect to a neighborhood function $\mathcal{N}$ and objective function c is denoted as $\mathcal{L}_{\mathcal{N}c}(\mathcal{S})$. A local optimum solution is not necessarily a global optimum solution s^*. To see this, let us consider the same instance of the multiprocessor scheduling problem described above. For a feasible schedule s, the objective function $c(s)$ gives the makespan of the schedule. Let us use the same neighborhood function defined before, i.e., $\mathcal{N}(s)$ includes all solutions that can be obtained from s by moving a single job from a machine with maximum load to one with minimum load. This neighborhood function is called the *jump* of *move* neighborhood [4]. Let the initial solution be as shown in Figure 18.3.

Note that since M_1 and M_2 have maximum load, this solution is local optimum as moving a single job cannot decrease the makespan. The makespan of this local optimum solution is 6, while the global optimum solution of Figure 18.2(b) has makespan 5.

Given a combinatorial optimization problem Π with instances $(\mathcal{S}, c)$ and a local search algorithm A that uses neighborhood function $\mathcal{N}$, we define the *locality gap* α_A of A as the largest possible ratio between the value of a local optimum solution and a global optimum one

$$\alpha_A = \underset{(\mathcal{S},c)\in\Pi}{\texttt{max}} \left\{ \underset{s\in\mathcal{L}_{\mathcal{N}c}(\mathcal{S})}{\texttt{max}} \left\{ \frac{c(s)}{c(s^*)}, \frac{c(s^*)}{c(s)} \right\} : c(s^*) = \underset{s'\in\mathcal{S}}{\texttt{optimum}}\, c(s') \right\} \tag{18.2}$$

Therefore, if an algorithm A can compute in polynomial time a local optimum solution for a combinatorial optimization problem Π with respect to a given neighborhood function, then A is an approximation algorithm for Π with approximation ratio α_A.

18.1.2 The Complexity of Computing Local Optimum Solutions

There is a large number of combinatorial optimization problems and natural neighborhood functions for them, for which we do not know any polynomial-time algorithm for computing local optimum solutions. There has been a lot of research on characterizing the class of problems that admit polynomial-time algorithms for finding local optimum solutions. One of the most notable works in this area is the research by Johnson et al. [5], who introduced the complexity class PLS of Polynomial-time Local Search problems.

This class includes all those problems and associated neighborhood functions which admit polynomial-time algorithms for deciding whether a given feasible solution is locally optimum and, if not, computes a better solution in its neighborhood. There is a reduction among problems in the class PLS which defines a subclass of complete PLS problems. It is unknown whether there exists polynomial-time algorithms for computing local optimum solutions for PLS-complete problems.

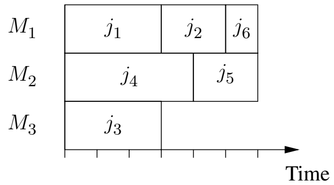


FIGURE 18.3 A local optimum solution with respect to the jump neighborhood and makespan objective function.

If we look closely at the IterativeImprovement algorithm described above, we note that its time complexity is dominated by the number of iterations of the while loop and by the time needed to search the neighborhood for a better solution. Given a combinatorial optimization problem, it is possible to scale the objective function c, so it yields integer values for all feasible solutions. Then, the IterativeImprovement algorithm will terminate in a pseudopolynomial number of iterations, since each iteration improves the value of the solution by an integral amount.

This observation led Orlin et al. [6] to study approximate locally optimum solutions: Given a value $\varepsilon > 0$, a solution s for an instance $(\mathcal{S}, c)$ of a combinatorial minimization problem is an *ε-local optimum* with respect to the neighborhood function $\mathcal{N}$ if

$$c(s) - c(s') \leq \varepsilon c(s') \quad \text{for all } s' \in \mathcal{N}(s)$$

Approximate locally optimum solutions can be defined in a similar manner for maximization problems. In Ref. [6] it is shown that every combinatorial optimization problem with a neighborhood function that can be efficiently searched has a fully polynomial-time algorithm for computing ε-local optimum solutions. This is a very interesting result, since as we show in Section 18.4, ε-local optimum solutions might be shown to be nearly global optimum.

18.1.3 Local Search in Approximation Algorithms

Despite its simplicity, local search has not been extensively used to design approximation algorithms. Among the reasons for this are that computing the locality gap of a local search algorithm is not easy, and for many problems natural local search algorithms have very large locality gaps leading to poor approximation algorithms. Consider, for example, the multiprocessor scheduling problem described above and the jump neighborhood function. Let us consider an instance of the problem, where J consists of an even number $n = km$ of jobs with unit processing times for some integer value $k > 0$, and let M consists of m machines. The solution shown in Figure 18.4 for this instance (half of the jobs are processed on machine M_1 and the other half on M_2) is a local optimum solution as moving a single job cannot decrease the makespan. An optimum solution, however, distributes the jobs evenly among all machines, and so it has makespan k. The locality gap of the IterativeImprovement algorithm with the jump neighborhood function is, then, at least $\frac{km/2}{k} = \frac{m}{2}$.

The problem with a local search algorithm based on the jump neighborhood is that it might get "trapped" in a local optimum solution of value far away from the optimum. If a neighborhood function is such that a local search algorithm always finds a global optimum solution, regardless of the initial solution, such a neighborhood function is called *exact*. For neighborhood functions that are not exact, getting trapped in a local optimum is a big problem, hence various local search techniques have been designed that can move a local search algorithm away from a local optimum solution. Variable-depth search, tabu search, simulated annealing, and genetic algorithms are among these techniques. In this chapter we consider only iterative improvement algorithms, as more complex local search techniques are much harder to analyze. In the sequel "local search" will mean iterative improvement.

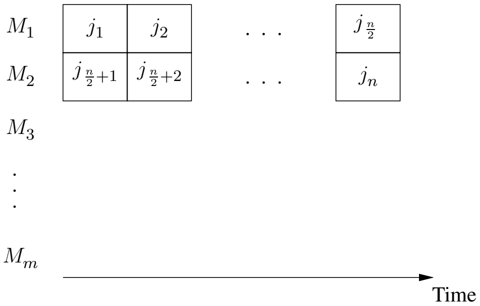


FIGURE 18.4 Instance showing a large locality gap for the jump neighborhood.

When designing an approximation algorithm using local search, if the locality gap of the algorithm with the selected neighborhood function is too large, we can try to improve it by selecting a different neighborhood function, and sometimes (as we show in the next section) by modifying the cost function.

Let us again consider the multiprocessor scheduling problem. Two other neighborhood functions that have been used for designing local search algorithms for this problem are the *swap* and *push* neighborhoods [4]. In the swap neighborhood, two jobs j_i and j_k on different machines are swapped (thus, interchanging their machine allocations). The push neighborhood allows moving a job j_i from its current machine M_j to a different one, M_h, and then it allows to recursively move from M_h those jobs with processing times smaller than p_i. It is not hard to find instances showing that, like the jump neighborhood, these two neighborhood functions have a large locality gap.

We could also define a *k-jump* neighborhood function, where up to k jobs are selected and moved to other machines. By selecting a sufficiently large value for k (e.g., by choosing $k = n$) we guarantee that the neighborhood function is exact and, therefore, that a local search algorithm based on such a neighborhood function will always find optimum solutions. The problem with this function is that given a solution s, the size of its neighborhood $|\mathcal{N}(s)|$ is exponential in k. Therefore, Step 3 of algorithm IterativeImprovement might require time that is exponential in k. The main challenge when designing a good local search approximation algorithm is to select a neighborhood function that yields small enough neighborhoods, so that deciding whether the current solution is a local optimum, or finding a better solution can be done efficiently, and such that the locality gap of the algorithm is small.

Computing the locality gap of a local search algorithm is, in general, not easy. We need to make use of the structural properties of local optimum solutions and relate them to the properties of global optimum solutions. In the next two sections we describe local search approximation algorithms for two NP-hard problems to illustrate this process. Sometimes it is a good idea to use Eq. (18.1) defining a local optimum solution to relate the value of local optimum and global optimum solutions. This idea is used in the last section of this chapter to design an approximation algorithm for the k-median problem.

18.2 An Approximation Algorithm for Multiprocessor Scheduling

As the example in Figure 18.4 shows, a local search algorithm for multiprocessor scheduling based on the jump neighborhood might return a solution that is much worse than the optimum. This happens when the algorithm gets trapped in a local optimum where several machines have the maximum load and the rest of them are idle. As in this case more than one machine has maximum load, moving a single job will not decrease the makespan of the schedule. However, by moving a job from a maximum load machine to an idle one, the number of machines with largest load will decrease. If we continue moving jobs away from machines with maximum load, eventually we might be able to decrease the makespan of the solution.

To force the IterativeImprovement algorithm to continue moving jobs away from machines with largest load, let us define a new objective function c' that assigns to every schedule s a pair $(c(s), d(s))$, where $c(s)$ is the makespan of s and $d(s)$ the number of machines with load $c(s)$. Given two solutions s and s', we let $c'(s) < c'(s)$ if $c(s) < c(s')$ or if $c(s) = c(s')$ and $d(s) < d(s')$.

Given a schedule s, its neighborhood $\mathcal{N}(s)$ has size at most $O(mn)$ since a neighbor of s can be obtained by moving any of the n jobs to any of the $m - 1$ machines which do not process it in s. Furthermore, we show below that the IterativeImprovement algorithm requires $O(n^2)$ iterations to find a local optimum solution. Thus, the time complexity of IterativeImprovement with the jump neighborhood is $O(mn^3)$.

We use the arguments presented in Ref. [7] to bound the number of iterations of the algorithm. Let $C_{\max}(i)$ and $C_{\min}(i)$ denote, respectively, the maximum and minimum machine loads at the beginning of the ith iteration. Let $\Delta(i) = C_{\max}(i) - C_{\min}(i)$. Since every iteration moves a job from a machine with maximum load to a machine with minimum load, then $C_{\max}(i)$ is a monotone nonincreasing function and $C_{\min}(i)$ a monotone nondecreasing function. Therefore, $\Delta(i)$ is also monotone nonincreasing.

We will bound the number of iterations by first bounding the maximum number of times that a job can be moved to other machines. Consider a job j_i that is moved to some machine M_k during the rth iteration. This means that $C_{\min}(r) = \ell(M_k, r)$, where $\ell(M_k, r)$ is the load of M_k at the beginning of the rth iteration. Then, assume that j_i is moved from M_k to some other machine at a later iteration $q > r$; thus, $C_{\max}(q) = \ell(M_k, q)$.

We need to consider two cases:

- Assume that no job is moved to machine M_k between iterations $r + 1$ and q. Then,

$$C_{\max}(q) = \ell(M_k, q) \leq \ell(M_k, r) + p_i = C_{\min}(r) + p_i \leq C_{\min}(q) + p_i$$

  The last inequality follows since $C_{\min}(i)$ is nondecreasing. Therefore,

$$p_i \geq C_{\max}(q) - C_{\min}(q) = \Delta(q)$$

  This implies that job j_i cannot be moved during the qth iteration since by moving j_i to a different machine the value of the objective function c' will not decrease.
- Therefore, at least one job needs to be moved to M_k during iterations $r+1, \ldots, q$. Let j_h be the last one of these jobs, and let j_h be moved to M_k during iteration u. Then, $C_{\min}(u) = \ell(M_k, u)$ and

$$C_{\max}(q) = \ell(M_k, q) \leq \ell(M_k, u) + p_h = C_{\min}(u) + p_h \leq C_{\min}(q) + p_h$$

  Hence,

$$p_h \geq C_{\max}(q) - C_{\min}(q) = \Delta(q)$$

  This implies that job j_h will not move during the following iterations.

From the above arguments, we conclude that a job j_i can move more than once, only if between any two consecutive moves at least one other job j_h gets fixed on some machine from which it will not move any further. Since at most $n - 1$ job movements can use j_h as the job that gets fixed, then the total number of job movements (and, thus, the total number of iterations of the algorithm) is $O(n^2)$.

Theorem 18.1

The IterativeImprovement algorithm with the jump neighborhood function and cost function c' always finds a schedule s with makespan at most $2 - \frac{1}{m}$ times the optimum one.

The proof of this theorem is essentially the same as Graham's proof for the performance ratio of his List scheduling algorithm [8]. Consider an instance $(\mathcal{S}, c)$ of the multiprocessor scheduling problem. The feasible set $\mathcal{S}$ is formed by all possible ways of scheduling the jobs J on the machines M. Let s^* be an optimum solution for this instance and let s be the local optimum solution computed by IterativeImprovement with the jump neighborhood and objective function c' (see Figure 18.5). Let M_i be a machine with maximum load $c(s)$ and let j_r be the last job processed by M_i.

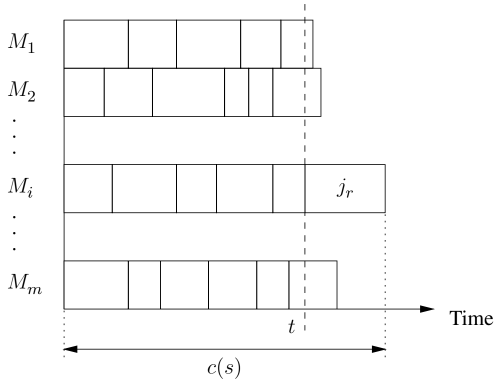


FIGURE 18.5 Solution computed by IterativeImprovement.

Let j_r start processing at time t, so $c(s) = t + p_r$. Since s is a local optimum solution, then every machine has load at least t. To see this, observe that if some machine M_k has load smaller than t, then moving j_r to M_k would decrease the cost $c'(s)$ of the solution. Since every machine's load is at least t, then

$$t \leq \frac{1}{m}\left(\sum_{j_i \in J} p_i - p_r\right) \leq c(s^*) - \frac{1}{m} p_r$$

as no schedule can process all jobs in J in time smaller than $\frac{1}{m}\sum_{j_i \in J} p_i$. Therefore,

$$\begin{aligned} c(s) = t + p_r &\leq c(s^*) - \frac{1}{m} p_r + p_r \\ &= c(s^*) + \left(1 - \frac{1}{m}\right) p_r \\ &\leq c(s^*) + \left(1 - \frac{1}{m}\right) c(s^*) \;\text{ as } p_r \leq c(s^*) \\ &= c(s^*)\left(2 - \frac{1}{m}\right) \end{aligned}$$

18.3 Finding Spanning Trees with Many Leaves

In this section we describe the local search algorithm of Lu and Ravi [9] for the *maximum leaves spanning tree problem*: Given an undirected graph $G = (V, E)$, find a spanning tree of G with maximum number of leaves. This problem is known to be NP-hard and MAX SNP-complete [10]. The algorithm of Lu and Ravi is simply the IterativeImprovement algorithm with the *exchange neighborhood function*, described below. The feasibility space $\mathcal{S}$ for this problem includes all spanning trees of the input graph G. For any spanning tree $s \in \mathcal{S}$, the objective function $c(s)$ gives the number of leaves in s.

Consider a spanning tree s. The exchange neighborhood of s is formed by all spanning trees that differ from s in a single edge, i.e.,

$$\mathcal{N}(s) = \{s' : s' \text{ is a spanning tree of G and } |s \cap s'| = n - 2\}$$

where n is the number of vertices of G. Here we view a spanning tree s as a collection of edges. Let $(u, v) \in E \setminus s$ be an edge that does not belong to the spanning tree s. Let s_{uv} be the unique path in s between vertices u and v. Note that if we add edge (u, v) to s we create a unique cycle, and by removing from this cycle any edge in s_{uv} we create a new spanning tree (see Figure 18.6). Since the path s_{uv} can have at most $n - 1$ edges, then the neighborhood $\mathcal{S}(s)$ of s has size at most $m(n - 1)$, where m is the number of edges in the graph. As this neighborhood function has polynomial size, Step 3 of algorithm IterativeImprovement can be performed in $O(mn)$ time.

Furthermore, each iteration of the algorithm increases the value of the solution s by at least 1. The initial solution s_0 must have at least 2 leaves, so $c(s_0) \geq 2$. Each iteration increases the value of the solution, and

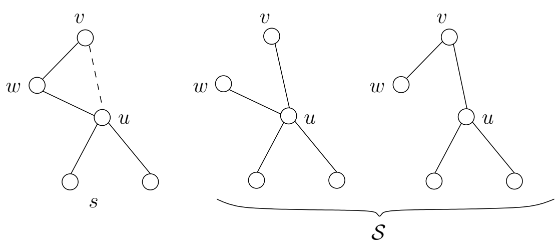


FIGURE 18.6 Neighborhood of spanning tree s. Edge (u, v) does not belong to s.

$c(s)$ cannot exceed $n-1$. Therefore, the maximum number of iterations that the while loop can perform is $n-3$. The time complexity of algorithm IterativeImprovement is, then, $O(mn^2)$. Now, let us compute the locality gap of the algorithm.

Theorem 18.2

The IterativeImprovement algorithm with the exchange neighborhood function has a locality gap of 10.

To prove the theorem we need to recall some basic properties of spanning trees. Let T be a spanning tree of a given graph $G = (V, E)$. A path in T containing only nodes of degree 2 in T is called a 2-*path.* For the rest of this section we use the convention that when referring to a tree T, the *degree* of a vertex u is the degree of u in T (not the degree of u in the graph G). So, for example, if u is a leaf of T we will say that the degree of u is 1.

Property 18.1

The number of nodes $N_3(T)$ of degree at least 3 in a spanning tree T of G is at most the number of leaves in T minus 2, i.e.,

$$N_3(T) \leq c(T) - 2$$

Proof

A spanning tree has maximum number of vertices of degree at least 3 if it is a full binary tree. In a full binary tree the number of leaves is one more than the number of internal nodes. Since in a full binary tree all internal nodes, except the root, have degree 3, then $c(T') = N_3(T') + 2$ for any binary tree T', and so

$$N_3(T) \leq c(T) - 2$$

for any spanning tree T of G. □

Property 18.2

The number $P_2(T)$ of 2-paths in any tree T is at most twice the number of leaves of T minus 3, i.e.,

$$P_2(T) \leq 2c(T) - 3$$

Proof

Let us choose a vertex of degree at least 3 as the root of T. Note that if the tree does not have any vertices of degree larger than 2 then T is a path, so $P_2(T) = 1$ and $c(T) = 2$.

A 2-path in T either connects a leaf or a vertex of degree at least 3 with its unique nearest ancestor of degree at least 3. Thus, every leaf and every node of degree at least 3 (except the root) has associated a unique 2-path, and so

$$P_2(T) \leq c(T) + N_3(T) - 1 \leq 2c(T) - 3$$ □

The last ingredient that we need to prove the theorem is the following lemma that relates the number of leaves in any spanning tree with the number of 2-paths in the local optimum tree s selected by the algorithm IterativeImprovement.

Lemma 18.1

Let T be a spanning tree of a graph $G = (V, E)$ and let s be the tree selected by IterativeImprovement. Let p be a 2-path of s. At most 4 vertices in p can be leaves in T.

Proof

We prove the lemma by contradiction. Assume that there is a 2-path p in s such that at least five of its vertices are leaves in T. Let w_1, w_2, w_3, w_4, and w_5 be five of the leaves of T that belong to p, as shown in Figure 18.7. Note that if two vertices u_i, u_j of p are not adjacent in p, then there cannot be an edge between them in G. This is because if such an edge exists, then adding (u_i, u_j) to s forms a unique cycle (see Figure 18.8); removing any edge of this cycle, other than (u_i, u_j), would increase the number of leaves

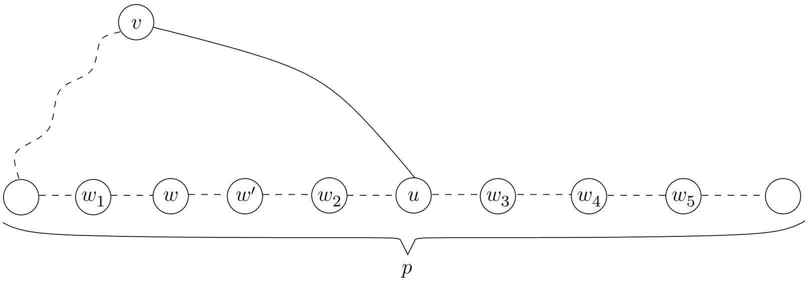


FIGURE 18.7 2-Path p of s. Edge (u, v) does not belong to s.

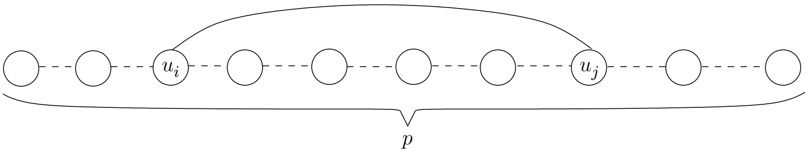


FIGURE 18.8 Cycle formed by the inclusion of edge (u_i, u_j) to p.

of s by 2, contradicting the assumption that s is a local optimum solution with respect to the exchange neighborhood.

Let us pick any vertex r not in p as the root of T. Let (u, v) be the first edge in the path from w_3 to r that does not belong to p (see Figure 18.7). Note that by the above argument, u and v cannot both belong to p. Without loss of generality let v not belong to p. Assume that the path s_{vw_3} in s from v to w_3 goes through vertices w_1 and w_2 (the other case, when such a path goes through w_4 and w_5 is similar). Adding edge (u, v) to s creates a unique cycle. Also, observe that u is an internal vertex in s, and thus, by adding (u, v) to s we decrease the number of leaves of s by at most 1. If we now remove from p any edge (w, w') in the path from w_1 to w_2 we create a new spanning tree s' in which w and w' are leaves, and so $c(s') \geq c(s) + 1$, contradicting the assumption that s is a local optimum solution. □

Now, we are ready to prove the theorem and to show that the IterativeImprovement algorithm with the exchange neighborhood has approximation ratio at most 10. Let s^* be a spanning tree of G with maximum number of leaves. Every leaf of s^* must be either

(a) a leaf of s; the total number of these leaves is at most $c(s)$,
(b) a vertex of degree at least 3 in s; there are at most $N_3(s) \leq c(s) - 2$ of these leaves by Property 18.1, or
(c) a vertex in a 2-path of s; by Lemma 18.1 and Property 18.2, the number of these leaves is at most $4P_2(s) \leq 8c(s) - 12$.

Combining (a)–(c), the number of leaves in s^* is

$$c(s^*) \leq c(s) + c(s) - 2 + 8c(s) - 12 \leq 10c(s)$$

Therefore,

$$\frac{c(s^*)}{c(s)} \leq 10$$

In Ref. [9] it is shown that by using more complex arguments, it can be proven that the locality gap of the algorithm is at most 5. Furthermore, by using a neighborhood function that allows the simultaneous exchange of two tree edges with two nontree edges, Lu and Ravi [9] show that the IterativeImprovement algorithm has locality gap 3.

18.4 Clustering Problems

In a clustering problem we are given a weighted graph $G = (V, E)$ and the goal is to partition the vertices of G into groups or clusters so that a certain objective function is optimized. Two classical clustering problems are the k-median problem and the facility location problem. In the k-median problem it is desired to partition the set of vertices V into k clusters, each with a distinguished vertex called a *center* or *median*, so that the sum of distances from the vertices to their clusters' medians is minimized.

Consider, for example, the weighted graph in Figure 18.9. If $k = 2$, then the two medians in an optimum solution are vertices 3 and 6. These two medians define a partition of the set of vertices, as each vertex must be in the same cluster as its nearest median (ties are broken arbitrarily); thus, the clusters are $\{1, 2, 3, 4, 8\}$ and $\{5, 6, 7, 9\}$. The sum of distances from the vertices to their nearest medians is $1+1+2+3+1+2+3 = 13$. A related problem is the *k-means* [11] problem where the goal is to minimize the sum of squared distances from the vertices to the medians.

In the facility location problem each vertex u of the graph has associated a cost $f(u)$ and the goal is to select a set F of vertices (and, thus, to partition G into $|F|$ clusters) so that the total cost $\sum_{u \in F} f(u)$ of the vertices in F plus the sum of distances from the vertices in $V \backslash F$ to F is minimized. Consider again the same graph of Figure 18.9. If the cost $f(u)$ of each vertex u is 3, then an optimum solution for the corresponding facility location problem is $F = \{3, 6\}$, and the cost of this solution is $3 + 3 + 13 = 19$.

Local search algorithms have been recently used to design good approximation algorithms for a variety of clustering problems including the k-median, k-means, and facility location problems [11–14]. In the next section we describe the algorithm of Arya et al. [15] for the metric version of the k-median problem.

18.4.1 Local Search Algorithm for the *k*-Median Problem

Consider a weighted graph $G = (V, E)$ in which the edge weight function $d : E \rightarrow \mathbb{R}$ satisfies the triangle inequality, i.e., for any three vertices $u, v, w \in V$, $d(u, v) + d(v, w) \geq d(u, w)$. The *metric k-median problem* is the k-median problem restricted to weighted graphs with weight functions that satisfy the triangle inequality.

Since a feasible solution for the k-median problem is a set $s \subseteq V$ of k vertices, the feasible set $\mathcal{S}$ consists of all subsets of k vertices. A natural way of defining the neighborhood of a solution s is through the use of the *swap operation.* A swap operation replaces one vertex in s with a vertex in $V \backslash s$, so $\mathcal{N}(s) = \{s' : s' = (s \backslash \{u\}) \cup \{v\}$ for every $u \in s$, $v \in V \backslash s\}$. For notational simplicity we denote the set $(s \backslash \{u\}) \cup \{v\}$ as $s - u + v$. The objective function $c(s) = \sum_{u \in V} d(u, s)$ gives the sum of distances from the vertices to the medians in the solution s. The distance $d(u, s)$ from a vertex u to the set s is defined as the distance from u to its closest median in the set s.

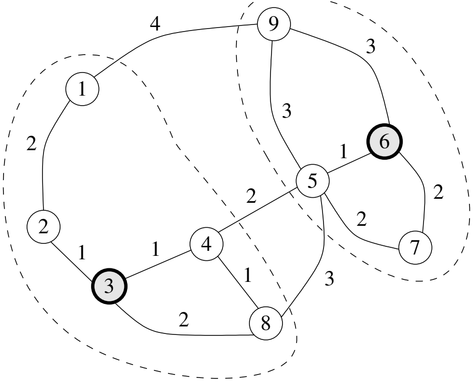


FIGURE 18.9 Clustering the vertices around vertices 3 and 6.

The size of the neighborhood $\mathcal{N}(s)$ of a solution s is $k(n-k)$. However, the number of iterations of the algorithm might not be polynomial in the size of the input. We show in the next section how to overcome this problem.

Theorem 18.3

The IterativeImprovement algorithm with the swap neighborhood has a locality gap of 5.

Let $s = \{s_1, s_2, \ldots, s_k\}$ be the solution computed by the algorithm. This set partitions the vertices V into k clusters $V_1, V_2, \ldots, V_k$, where all vertices in cluster V_i are closer to s_i than to any other median in s (ties are broken arbitrarily). Let $s^* = \{s_1^*, s_2^*, \ldots, s_k^*\}$ be an optimum solution, and let $V_1^*, V_2^*, \ldots, V_k^*$ be the partition induced by s^*. For any vertex $u \in V$ let $V_{\sigma(u)}$ denote its cluster in s and let $s_{\sigma(u)}$ be the median in cluster $V_{\sigma(u)}$. Then, the value $c(s)$ of solution s is $c(s) = \sum_{u \in V} d(u, s_{\sigma(u)})$.

Note that since s is a local optimum solution, then no swap operation can improve its value, i.e., for any pair of vertices $s_i \in s$, $v \in V \backslash s$,

$$c(s - s_i + v) \geq c(s) \tag{18.3}$$

To prove Theorem 18.3, first we will pair each median $s_i^* \in s^*$ with some median $\rho(s_i^*) \in s$ in such a way that no vertex in s is paired with more than two vertices of s^*. The pairing relation ρ is specified below. By inequality (18.3), for every vertex $s_i^* \in s^*$,

$$c(s - \rho(s_i^*) + s_i^*) - c(s) \geq 0 \tag{18.4}$$

and this inequality holds regardless of whether s_i^* is in s or not.

Observe that $c(s - \rho(s_i^*) + s_i^*)$ and $c(s)$ differ only by the contributions made by those vertices in $\rho(s_i^*)$'s cluster and s_i^*'s cluster to the values of the two solutions, as shown in Figure 18.10. Therefore,

$$0 \leq c(s - \rho(s_i^*) + s_i^*) - c(s) \leq \sum_{u \in V_i^*} (d(u, s^*) - d(u, s)) + \sum_{v \in V_{\sigma(\rho(s_i^*))} \backslash V_i^*} (d(v, s - \rho(s_i^*) + s_i^*) - d(v, s)) \tag{18.5}$$

Adding inequalities (18.5) over all medians $s_i^* \in s^*$, we get

$$\begin{aligned} 0 &\leq \sum_{s_i^* \in s^*} (c(s - \rho(s_i^*) + s_i^*) - c(s)) \\ &\leq \sum_{s_i^* \in s^*} \sum_{u \in V_i^*} (d(u, s^*) - d(u, s)) + \sum_{s_i^* \in s^*} \sum_{v \in V_{\sigma(\rho(s_i^*))} \backslash V_i^*} (d(v, s - \rho(s_i^*) + s_i^*) - d(v, s)) \end{aligned} \tag{18.6}$$

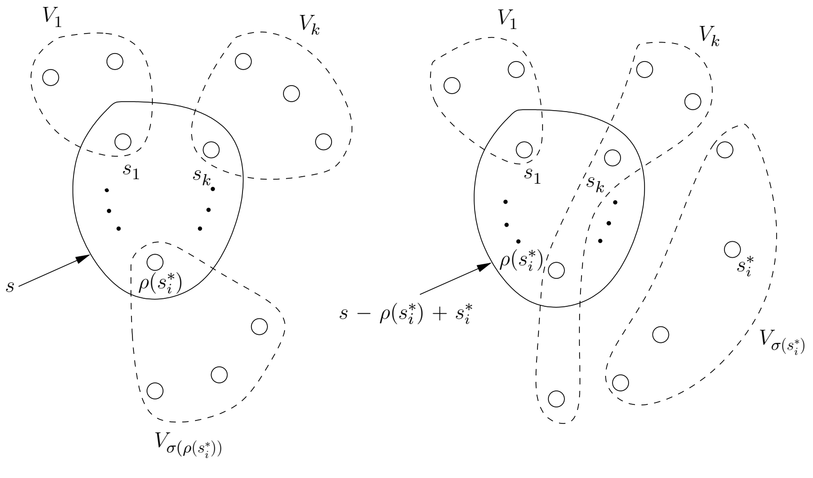


FIGURE 18.10 Solutions s and $s - \rho(s_i^*) + s_i^*$.

Let us look closely at the first term on the right-hand side of this last inequality. Since $\cup_{i=1}^{k} V_i^* = V$, then

$$\sum_{s_i^* \in s^*} \sum_{u \in V_i^*} (d(u, s^*) - d(u, s)) = \sum_{u \in V} d(u, s^*) - \sum_{u \in V} d(u, s) = c(s^*) - c(s) \tag{18.7}$$

The second term in the last inequality of (18.6) is more complicated, so we will spend most of the rest of this section showing that

$$\sum_{s_i^* \in s^*} \sum_{v \in V_{\sigma(\rho(s_i^*))} \setminus V_i^*} (d(v, s - \rho(s_i^*) + s_i^*) - d(v, s)) \leq 4c(s^*) \tag{18.8}$$

Combining Eqs. (18.6)–(18.8) we get

$$0 \leq c(s^*) - c(s) + 4c(s^*) = 5c(s^*) - c(s)$$

From which it follows that

$$\frac{c(s)}{c(s^*)} \leq 5$$

as Theorem 18.3 claims. It just remains to prove inequality (18.8). The proof in Ref. [15] for the validity of this inequality is an elegant argument that exploits the fact that the edge weights satisfy the triangle inequality. First, let us define the relation ρ. We partition the set s of k medians into three groups, A_s, B_s, and C_s, as follows:

$$\begin{aligned} A_s &= \{s_i \in s : |V_i \cap V_j^*| \leq \frac{1}{2}|V_j^*| \text{ for all clusters } V_j^*,\ 1 \leq j \leq k\} \\ B_s &= \{s_i \in s : |V_i \cap V_j^*| > \frac{1}{2}|V_j^*| \text{ for exactly one cluster } V_j^*,\ 1 \leq j \leq k\} \\ C_s &= s \backslash (A_s \cup B_s) \end{aligned}$$

Observe that set B_s associates exactly one vertex s_j^* with each vertex $s_i \in B_s$. For each $s_i \in B_s$, we define the relation ρ as follows:

$$\rho(s_j^*) = s_i, \quad \text{where } s_j^* \text{ is such that } |V_i \cap V_j^*| > \frac{1}{2}|V_j^*|$$

For the remaining $|s| - |B_s|$ medians in s^*, the relation ρ associates them to vertices in A_s. Vertices in C_s are not mapped to any of the medians in the optimum solution. Note that for every vertex $s_i \in C_s$ there are at least two medians s_j^*, s_ℓ^* whose clusters share at least half of their vertices with s_i's cluster. Therefore, $|C_s| < \frac{1}{2}(|s| - |B_s|)$ and so $|A_s| \geq \frac{1}{2}(|s| - |B_s|)$.

Let $A_s = \{s_0, s_1, \ldots, s_{|A_s|-1}\}$ and let the, still, unmapped medians in s^* be $U^* = \{s_0^*, s_i^*, \ldots, s_{|s|-|B_s|-1}^*\}$. Then, for all $s_j^* \in U^*$ we define

$$\rho(s_j^*) = s_{j \bmod |A_s|}$$

Note that at most two medians of s^* are mapped to the same vertex of s.

Consider a pair $(\rho(s_i^*), s_i^*)$, and the swap operation that exchanges these two vertices transforming the solution s into $s - \rho(s_i^*) + s_i^*$. As Figure 18.10 shows, this swap modifies some of the clusters by reassigning some of the vertices in $V_{\sigma(\rho(s_i^*))}$ and V_i^*. The change in the cost of the solution (see inequality [18.5]) is

$$\sum_{u \in V_i^*} (d(u, s^*) - d(u, s)) + \sum_{v \in V_{\sigma(\rho(s_i^*))} \setminus V_i^*} (d(v, s - \rho(s_i^*) + s_i^*) - d(v, s))$$

To find an upper bound for this value, we define a 1-1 and onto function $\pi : V \rightarrow V$ that for every median $s_i^* \in s^*$ maps vertices in its cluster V_i^* to V_i^*. The function is defined as follows. Fix a vertex $s_i^* \in s^*$. Sort the vertices in cluster V_i^* so that vertices in $V_i^* \cap V_j$ appear before vertices in $V_i^* \cap V_\ell$ for all

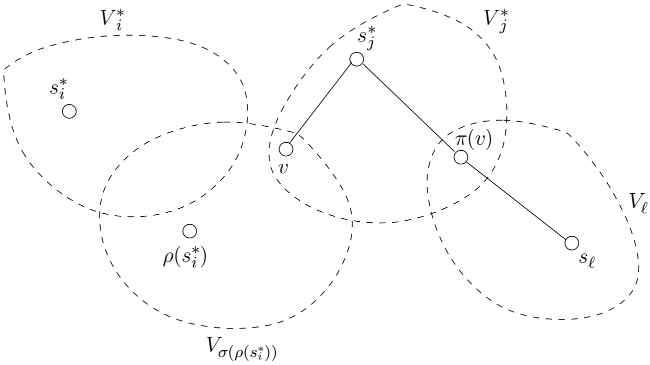


FIGURE 18.11 Reassigning vertices from $V_{\sigma(\rho(s_i^*))}$.

$1 \le j < \ell \le k$. Index the vertices in V_i^* in this order starting at 0. Let the vertices indexed in this manner be $v^*_{i0}, v^*_{i1}, \ldots, v^*_{i(p-1)}$, where $p = |V_i^*|$. Now, define

$$\pi(v^*_{ij}) = v^*_{ir}, \quad \text{where } r = \left(i + \left\lfloor \frac{p}{2} \right\rfloor\right) \bmod p, \quad \text{for all } 0 \le j \le d-1$$

Note that even when we do not know the optimum solution s^*, we know that such solution exists and, thus, this function also exists. Consider a vertex $v \in V_{\sigma(\rho(s_i^*))} \setminus V_i^*$ as depicted in Figure 18.11. Let v belong to cluster V_j^* in the optimum solution. Since $|V_{\sigma(\rho(s_i^*))} \cap V_j^*| \le \frac{1}{2}|V_j^*|$, then by the definition of π, we know that $\pi(v) \notin V_{\sigma(\rho(s_i^*))}$ and, thus, in solution s this vertex $\pi(v)$ belongs to some cluster $V_\ell \ne V_{\sigma(\rho(s_i^*))}$.

Since the swap operation that exchanges $\rho(s_i^*)$ and s_i^*, removes only one vertex, $\rho(s_i^*)$, from s, then $s_\ell \in s - \rho(s_i^*) + s_i^*$. Therefore,

$$d(v, s - \rho(s_i^*) + s_i^*) \le d(v, s_\ell)$$

Then, by the triangle inequality (see Figure 18.11),

$$\begin{aligned} d(v, s - \rho(s_i^*) + s_i^*) &\le d(v, s_\ell) \\ &\le d(v, s_j^*) + d(s_j^*, \pi(v)) + d(\pi(v), s_\ell) \\ &= d(v, s^*) + d(\pi(v), s^*) + d(\pi(v), s) \end{aligned}$$

Adding these inequalities for all vertices s_i^* as required on the left-hand side of Eq. (18.8) yields

$$\begin{aligned} &\sum_{s_i^* \in s^*} \sum_{v \in V_{\sigma(\rho(s_i^*))} \setminus V_i^*} (d(v, s - \rho(s_i^*) + s_i^*) - d(v, s)) \\ &\quad \le \sum_{s_i^* \in s^*} \sum_{v \in V_{\sigma(\rho(s_i^*))} \setminus V_i^*} ((d(v, s^*) + d(\pi(v), s^*) \\ &\quad + d(\pi(v), s) - d(v, s)) \end{aligned}$$

Since $\cup_{s_i^* \in s^*} (V_{\sigma(\rho(s_i^*))} \setminus V_i^*) \subseteq V$ and ρ associates at most two vertices from s^* with any vertex in s, then

$$\begin{aligned} &\sum_{s_i^* \in s^*} \sum_{v \in V_{\sigma(\rho(s_i^*))} \setminus V_i^*} (d(v, s^*) + d(\pi(v), s^*) + d(\pi(v), s) - d(v, s)) \\ &\quad \le 2 \sum_{v \in V} (d(v, s^*) + d(\pi(v), s^*) \\ &\quad + d(\pi(v), s) - d(v, s)) \end{aligned}$$

Recall that the function π is 1-1 and onto, therefore,

$$\sum_{v\in V} d(v, s^*) = \sum_{v\in V} d(\pi(v), s^*) = c(s^*) \text{ and}$$
$$\sum_{v\in V} d(\pi(v), s) = \sum_{v\in V} d(v, s) = c(s)$$

Combining these inequalities yields

$$\sum_{s_i^*\in s^*} \sum_{v\in V_{\sigma(\rho(s_i^*))}\setminus V_i^*} (d(v, s - \rho(s_i^*) + s_i^*) - d(v, s)) \le 4c(s^*)$$

as required.

18.4.2 Approximate Local Optimum Solutions

Even when the locality gap of the above algorithm is at most 5, it is not an approximation algorithm, as the number of iterations needed by IterativeImprovement to find a local optimum solution might be superpolynomial. This is because the algorithm might potentially select as intermediate solutions a large fraction of all the subsets of k vertices in the input graph G.

To fix this problem we can change the stopping condition of the IterativeImprovement algorithm so it finishes as soon as it finds an approximate local optimum solution, i.e., when the solution s is such that every neighboring solution $s' \in \mathcal{N}(s)$ has value

$$c(s') > (1-\varepsilon)c(s)$$

for some accuracy $\varepsilon > 0$. By changing in this manner the condition of the while loop in Step 2 of the algorithm, we ensure that in each iteration the value of the solution decreases by at least a factor of $1-\varepsilon$. Therefore, the maximum number of iterations is $\log(c(s_0)/c(s^*))/\log(\frac{1}{1-\varepsilon})$, which is polynomial in the size of the input; s_0 is the initial solution selected in Step 1.

If we change the algorithm as described above, the analysis needs to change also, since now condition (18.4) does not hold, but the following one does:

$$c(s - \rho(s_i^*) + s_i^*) \ge (1-\varepsilon)c(s) \quad \text{for all } s_i^* \in s^*$$

Using this inequality in our analysis gives only a slight worsening in the locality gap (and, thus on the approximation ratio) of the algorithm, as now

$$\frac{c(s)}{c(s^*)} \le \frac{5}{1-3\varepsilon}$$

Arya et al. [15] show that by using the *p-swap* neighborhood which puts in the neighborhood $\mathcal{N}(s)$ of a solution s any subset s' of k vertices that differs from s in at most p vertices, the IterativeImprovement algorithms has locality gap $(3 + 2/p)$. To the date when this paper was written this was the best known approximation algorithm for the metric k-median problem.

References

[1] Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, New York, 1982.

[2] Aarts, E. H. L. and Lenstra, J. K., Eds., *Local Search in Combinatorial Optimization*, Wiley, Chichester, 1997.

[3] Osman I. H. and Kelly, J. P., Eds., *Metaheuristics: Theory and Applications*, Kluwer, Boston, 1996.

[4] Schuurman, P. and Vredeveld, T., Performance guarantees of local search for multiprocesor scheduling, *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science, Vol. 2081, Springer, Berlin, 2001, p. 370.

[5] Johnson, D. S., Papadimitriou, C. H., and Yannakakis, M., How easy is local search? *JCSS*, 37, 79, 1988.

[6] Orlin, J. B., Punnen, A. P., and Schulz, A. S., Approximate local search in combinatorial optimization, *SIAM J. Comput.*, 33(5), 1201, 2004.

[7] Brucker, P., Hurink, J., and Werner, F., Improving local search heuristics for some scheduling problems II, *Discrete Appl. Math.*, 72, 47, 1997.

[8] Graham, R. L., Bounds for certain multiprocessor anomalies, *Bell Syst. Tech. J.*, 45, 1563, 1966.

[9] Lu, H. and Ravi, R., The power of local optimization: approximation algorithms for maximum-leaf spanning tree, *Proc. 13th Annual Allerton Conf. on Communication, Control, and Computing*; 1992, p. 533.

[10] Galbiati, G., Maffioli, F., and Morzenti, A., A short note on the approximability of the maximum leaves spanning tree problem, *Inf. Process. Lett.*, 52, 49, 1994.

[11] Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y., A local search approximation algorithm for k-means clustering, *Comput. Geometry: Theor. Appl.*, 28, 89, 2004.

[12] Charikar, M. and Guha, S., Improved combinatorial algorithms for the facility location and k-median problems, *Proc. of FOCS,* 1999, p. 378.

[13] Chudak, F. and Williamson, D., Improved approximation algorithms for capacitated facility location problems, *Proc. 7th Conf. on Integer Programming and Combinatorial Optimization,* 1999, p. 9.

[14] Korupolu, M., Plaxton, C. G., and Rajaraman, R., Analysis of a local search heuristic for facility location problems, *J. Algorithms*, 37, 237, 2000.

[15] Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., and Pandit, V., Local search heuristics for k-median and facility location problems, *SIAM J. Comput.*, 33(3), 544, 2004.

19
Stochastic Local Search

Holger H. Hoos
University of British Columbia

Thomas Stützle
Free University of Brussels

19.1 Introduction

Stochastic local search (SLS) algorithms are among the most successful techniques for solving computationally hard problems from computing science, operations research and various application areas; examples range from propositional satisfiability, routing and scheduling problems to genome sequence assembly, protein structure prediction and winner determination in combinatorial auctions. Because of their versatility and excellent performance in combination with the fact that efficient implementations can often be achieved relatively easily, SLS methods enjoy an ever-increasing popularity among researchers and practitioners.

Local search techniques have a long history; they range from simple constructive and iterative improvement algorithms to rather complex methods that require significant fine-tuning, such as evolutionary algorithms (EAs) or simulated annealing (SA). The key idea behind local search is to iteratively expand or improve a current candidate solution by means of small modifications. Most local search algorithms make use of randomised decisions, for example, in the context of generating initial solutions or when determining search steps, and are therefore referred to as *stochastic local search algorithms.* (It may be noted that formally, deterministic local search algorithms can be seen as special cases of SLS algorithms, since deterministic decisions can be modelled using degenerate probability distributions.) To define an SLS algorithm, the following components have to be specified. (A formal definition can be found in Chapter 1 of the book by Hoos and Stützle [1].)

Search space: the set of *candidate solutions* (or *search positions*) for the given problem instance; candidate solutions typically consist of a number of discrete *solution components.*

Solution set: specifies all search positions that are considered to be (feasible) solutions of the given problem instance.

Neighbourhood relation: specifies the direct neighbours of each candidate solution s, i.e., the search positions that can be reached from s in one search step.

Memory states: used to hold information about the search mechanism beyond the search position (e.g., tabu tenure of solution components in tabu search (TS), or temperature in SA); there may be only a single, constant state in the case of algorithms that do not use memory, such as simple iterative improvement.

Initialisation function: specifies search initialisation in the form of a probability distribution over initial search positions and memory states.

```
Stochastic Local Search:
    determine initial search state
    While termination criterion is not satisfied:
        perform search step
        if necessary, update incumbent solution
    return incumbent solution or report failure
```

FIGURE 19.1 General outline of an SLS algorithm.

Step function: determines the computation of search steps by mapping each search position and memory state to a probability distribution over its neighbouring search positions and memory states.

Termination predicate: used to decide search termination based on the current search position and memory state.

Based on these components, SLS algorithms work as illustrated in Figure 19.1. When applied to optimisation problems, whose definition comprises an objective function that specifies the quality of solutions, SLS algorithms need to keep track of the best solution encountered during the search process, the so-called *incumbent solution*. For decision problems, the search process is usually terminated as soon as a solution for the given problem instance is found, and repeated updating of an incumbent solution is not required.

Among the components underlying any SLS algorithm, the neighbourhood relation and the step function are particularly important. Typically, neighbourhood relations have to be defined in a problem-specific way, and it is often difficult to predict which of the various choices that can be made in this context will result in the best performance. However, standard types of neighbourhood relations exist; particularly widely used are the so-called *k-exchange neighbourhoods*, in which two candidate solutions are direct neighbours if and only if they differ in at most k solution components. As an example, consider the 2-exchange neighbourhood for the (symmetric) travelling salesman problem (TSP), under which two candidate solutions are direct neighbours if one can be obtained from the other by replacing two edges of s by two alternate edges (see Figure 19.2). Every neighbourhood relation induces a *neighbourhood graph*, whose vertex set corresponds to the given search space, and in which each pair of neighbouring search positions is connected by an edge. Many important properties of the neighbourhood relation are reflected in the neighbourhood graph; for example, k-exchange neighbourhoods induce symmetric, k-regular neighbourhood graphs.

The step function defines how the search process moves from one search state to the next, where a *search state* is a combination of a search position and a memory state. Search steps are usually defined by means of a procedure that draws a sample from the probability distribution determined by the underlying step function; similar procedural specifications are used for the initialisation function and termination predicate. The search process is typically guided by an *evaluation function* that is used to heuristically assess or rank candidate solutions. For combinatorial optimisation problems, the objective function (which is part of the problem definition) is often also used as an evaluation function. In the case of combinatorial decision problems, however, there can be considerable freedom in choosing an appropriate evaluation function. Furthermore, some SLS methods use multiple evaluation functions or modify the evaluation function while searching. (For this reason, we do not explicitly specify a single evaluation function as a component of our formal definition of an SLS algorithm.)

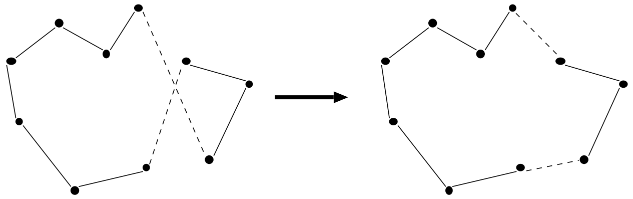


FIGURE 19.2 Example of a 2-exchange step for the symmetric TSP.

The concept of SLS, as defined above, provides a unifying framework for many different types of algorithms. In particular, it captures constructive algorithms, whose search space contains partial solutions of a given problem, i.e., candidate solutions that can be extended by adding solution components (such as edges in the case of the TSP). Consequently, search methods that are strongly based on constructive search procedures, such as greedy randomised adaptive search procedures (GRASP) [2,3] or ant colony optimisation (ACO) [4,5], fall under the general SLS framework. Similarly, population-based algorithms, which operate on ensembles of candidate solutions, are covered by our definition by considering search positions to consist of sets of individual candidate solutions. Note that in this case, step functions can be defined to model operations on populations, such as recombination in the case of EAs (see also Chapter 2 of Hoos and Stützle's book [1]).

Many SLS algorithms are based on generic methods for controlling and directing a simpler, subsidiary search process (such as an iterative improvement procedure, see Section 19.2), and substantial research efforts have been focused on the development and study of such general SLS methods, which are also commonly known as *metaheuristics* [6]. While high-level search strategies and mechanisms provide an important basis for developing SLS algorithms for a broad range of problems, it is important to keep in mind that other aspects, including the choice of the search space and neighbourhood relation as well as techniques for the efficient implementation of local search steps, are also crucial ingredients of high-performance SLS algorithms. Research in the area of SLS comprises all aspects of the design, implementation and analysis of SLS algorithms.

The remainder of this chapter provides an overview of widely used SLS methods for discrete combinatorial problems, including iterative improvement techniques (Section 19.2); so-called "simple" SLS methods (Section 19.3), including SA and tabu search, as well as the less widely known method of dynamic local search (DLS); hybrid SLS methods (Section 19.4), such as iterated local search (ILS) and GRASP; and population-based SLS methods (Section 19.5), in particular, ACO and EAs. The chapter closes with a brief discussion of some general issues and interesting research directions in the area of SLS.

19.2 Iterative Improvement

One of the most basic SLS methods is based on the idea of iteratively improving a candidate solution of the given problem with respect to an evaluation function. More precisely, the search is started from some initial position, and in each search step, the current candidate solution s is replaced with a neighbouring candidate solution s' with better evaluation function value. The search is terminated when a *local minimum* is reached, i.e., a candidate solution s with $g(s) \leq g(s')$ for all direct neighbours s' of s, where g is the evaluation function that is to be minimised. This SLS method is called *iterative improvement*; it is also known as *iterative descent* or *hill climbing* (the latter is motivated by an equivalent formulation where a given evaluation function is to be maximised).

Iterative improvement algorithms find local optima of a given evaluation function. Since local optima are defined with respect to a given neighbourhood relation N, it is quite obvious that the choice of N is of crucial importance for the performance of any iterative improvement procedure. While the use of larger neighbourhoods results in better quality local optima, the time complexity of determining improving search steps increases with neighbourhood size. For example, in the commonly used k-exchange neighbourhoods, each search position has $\mathcal{O}(n^k)$ direct neighbours (where n is the number of solution components in each candidate solution), i.e., the neighbourhood size is exponential in k, and the same holds for the time required for identifying improving neighbours in the worst case. This leads to a general trade-off between solution quality and run time of iterative improvement algorithms. In practice, search steps with quadratic or cubic time complexity can already lead to prohibitively high computation times when solving large problem instances.

The time complexity of local search steps can be significantly reduced using two generic techniques. Firstly, *caching and incremental updating techniques* can be used to significantly reduce the often considerable cost of computing from scratch the evaluation function values of all neighbours of the current search position in each search step. Instead, these values are stored and updated based on the actual effects of each search step (see also Chapter 1 of the book by Hoos and Stützle [1]). Secondly, the size of large neighbourhoods can be substantially reduced by excluding from consideration neighbours of the current candidate solution that are provably or likely nonimproving. Such *neighbourhood pruning techniques* have a long history; they include fixed radius searches, nearest neighbour lists, the use of so-called "don't look bits" in the context of the TSP [7], and reduced neighbourhoods in the case of the job-shop scheduling problem [8]. These techniques are essential for the design of high-performance SLS algorithms based on large neighbourhoods, but they can also lead to significant performance improvements when used in combination with smaller neighbourhoods.

Another factor that has a significant influence on the speed and performance of an iterative improvement algorithm is the mechanism used for determining the search steps—the so-called *pivoting rule* [9]. The two most widely used pivoting rules are best improvement and first improvement. *Iterative best improvement* chooses in each step one of the neighbouring search positions that results in a maximal possible improvement of the evaluation function value; ties can be broken randomly, based on the order in which the neighbourhood is examined or by using secondary criteria. *Iterative first improvement* examines the neighbourhood in some predefined order and performs the first improving search step encountered during this inspection. Clearly, the local optima found by this method depend on the order in which the neighbourhood is scanned. Instead of using predefined, fixed orders, it can be beneficial to examine the neighbourhood in random order, and repeated runs of such random-order first-improvement algorithms are often able to identify many different local optima, even when started from the same initial position [1].

Iterative first improvement usually requires more steps than iterative best improvement to reach local optima of comparable quality, but the individual improvement steps are typically found much faster, since in many cases, they do not require inspection of the entire neighbourhood. However, best-improvement algorithms often benefit more significantly from caching and incremental updating strategies, and as a result, are not always slower than first-improvement algorithms in reaching a local minimum. It may also be noted that to identify a candidate solution as a local optimum both, first- and best-improvement algorithms need to inspect the entire local neighbourhood; the time required for this final check (the so-called check-out time) can be reduced using "don't look bits" [7,10].

An interesting way of achieving a good trade-off between neighbourhood size and time complexity of local search steps is to use multiple neighbourhood relations. *Variable neighbourhood descent* (VND), a variant of a more general SLS method called variable neighbourhood search (VNS) [11,12], is an iterative improvement method that switches systematically between several neighbourhood relations N_1, N_2, ..., N_k, which are typically ordered according to increasing size. Starting from an initial candidate solution, VND performs iterative improvement using N_1. Once a local optimum w.r.t. N_1 is found, the search is continued in N_2. Generally, whenever a local optimum of N_i has been found, VND switches to N_{i+1}. However, as soon as an improvement has been achieved in any neighbourhood N_i (with $i > 1$), the search is continued using N_1. The idea underlying this mechanism is to use small neighbourhoods (which can be searched most efficiently) whenever possible. Variable neighbourhood descent terminates when a local optimum of N_k has been found. Empirical results have shown that VND algorithms are often significantly more efficient in finding high-quality local optima than simple iterative improvement algorithms using large neighbourhoods. It may be noted that other variants of the more general approach of variable neighbourhood search, such as basic VNS or skewed VNS [11,12], are conceptually more closely related to ILS (see Section 19.4).

Finally, a number of SLS algorithms use *very large-scale neighbourhoods* [13], whose size is often exponential in the size of the given problem instance. This type of neighbourhood usually has to be searched heuristically, which is the case in variable-depth search [14,15] and ejection chain algorithms [16]. However, there are specially structured neighbourhoods that can be searched exactly and efficiently using network-flow techniques or dynamic programming [17–21]. For an overview of these techniques we refer to Chapter 20 of this book.

19.3 “Simple” SLS Methods

Iterative improvement algorithms terminate when they reach a local optimum of the given evaluation function. In the following, we discuss several approaches that allow SLS algorithms to escape from local optima by occasionally accepting worsening search steps. The methods discussed in this section are “simple” in the sense that they are usually based on a single, fixed neighbourhood relation.

Randomised Iterative Improvement

One of the simplest ways to allow worsening search steps is to occasionally move to a randomly chosen neighbouring search position. In randomised iterative improvement (RII), this idea is implemented by switching probabilistically between two types of search steps within the same neighbourhood structure: iterative improvement steps and so-called *uninformed random walk steps*, in which a neighbour is chosen uniformly at random. More precisely, in each search step, an uninformed random walk step is performed with probability w_p, and an iterative improvement step is performed otherwise (i.e., with probability $1 - w_p$). The parameter w_p is called *walk probability* or *noise parameter*. Using this mechanism, arbitrarily long sequences of random walk steps can be performed, where the probability of r consecutive random walk steps is w_p^r. Extending an iterative improvement algorithm into an RII algorithm typically requires only few lines of code and introduces only one parameter. Despite their conceptual simplicity, RII algorithms can perform quite well; for example, they have provided the basis for GSAT with random walk, a well-known algorithm for the propositional satisfiability problem (SAT) [22]. However, there are relatively few RII algorithms, perhaps because more complex SLS algorithms often achieve better performance.

Probabilistic Iterative Improvement

Random walk steps, as performed in RII, may lead to significant deteriorations in evaluation function value. An attractive alternative is to base the acceptance of a worsening search step on the change in evaluation function caused by it; this is the key idea behind probabilistic iterative improvement (PII). At each step, PII selects a neighbouring candidate solution according to a function $p(g, s)$, which determines a probability distribution over the neighbourhood of S taking into account the evaluation function g. In practice, samples of this probability distribution can be taken as follows: first, a neighbour s' of the current search position s is selected uniformly at random; then, a decision is made whether s' is accepted as the new current search position. When minimising g, the probability of accepting s', $p_{accept}(T, s, s')$, is in many cases based on the following probability function:

$$p_{accept}(T, s, s') := \begin{cases} 1 & \text{if } g(s') < g(s) \\ \exp\left(\frac{g(s)-g(s')}{T}\right) & \text{otherwise} \end{cases} \tag{19.1}$$

where T is a parameter that determines the degree to which worsening search steps are likely to be accepted. This so-called *Metropolis condition* is frequently used in SA algorithms (discussed in the following), and in this context, the parameter T is called *temperature*. In fact, PII with the Metropolis condition is equivalent to constant-temperature SA, which has been shown to perform well if suitable temperature values are used [23,24].

Simulated Annealing

A natural generalisation of PII with the Metropolis condition is obtained by modifying the parameter T during the search; for example, by gradually decreasing T, the search process becomes increasingly greedy. This is the key idea underlying SA, an SLS method that is inspired by the annealing process of solids, and which has been independently proposed by Kirkpatrick et al. [25] and Cerný [26].

In each step of a standard SA algorithm, first, a neighbour s' of the current candidate solution s is chosen using a *proposal mechanism* (in the simplest case, this can be a uniform random choice from the local neighbourhood of s); next, a parameterised probabilistic *acceptance criterion* (e.g., the Metropolis condition—see Eq. (19.1)) is used to decide whether to perform a search step from s to s'. The way in which the temperature parameter T is modified during the search process is determined by the *annealing schedule*, a function that defines for each search step i a temperature value $T(i)$. Annealing schedules are usually specified in the form of an initial temperature T_0, the number of search steps performed at each temperature value (this is often chosen as a multiple of the neighbourhood size) and a temperature update scheme (e.g., geometric cooling according to $T_{k+1} := \alpha \cdot T_k$, where α is a parameter between 0 and 1). Obviously, the performance of an SA algorithm is highly dependent on its annealing schedule. SA algorithms frequently use special termination conditions, such as the *acceptance ratio* (i.e., the ratio of accepted versus proposed search steps) or the number of subsequent temperature values that have been used since the last improvement in the incumbent solution has been achieved.

Simulated annealing is one of the earliest and most prominent generic SLS methods. SA algorithms have been applied to a wide range of combinatorial problems; they have also been intensely studied analytically (giving rise to some interesting results, such as proofs regarding the convergence to optimal solutions) and experimentally. Many variants of SA have been proposed and studied, such as threshold accepting, where the probabilistic acceptance criterion is replaced by a deterministic mechanism [27]. For a more detailed discussion on SA we refer to Chapter 25 of this book.

Tabu Search

Differently from the previously discussed SLS methods, TS makes direct and systematic use of memory to guide the search process [28]. The key idea behind *simple* TS, the most basic TS variant, is to use short-term memory to prevent a subsidiary iterative improvement procedure from returning to recently visited search positions, which allows the search process to escape from local optima of the given evaluation function. When performing iterative improvement steps, only *permissible neighbours* of the current candidate solution are considered, i.e., neighbouring search positions that are not declared tabu. The tabu memory is updated after each search step, such that search steps cannot be undone for a fixed amount of time (*tabu tenure*). Typically, the tabu mechanism is implemented by storing with each solution component the time (in terms of step number) when it was last used in the current candidate solution, and to only permit solution components to be introduced by a search step for which the difference between this time stamp and the current time exceeds the tabu tenure. For example, in a simple TS algorithm based on the 2-exchange neighbourhood for the TSP, the edges removed in a 2-exchange step may not be reintroduced for *tt* subsequent search steps, where *tt* is the tabu tenure parameter. Many simple TS algorithms use an *aspiration criterion*, which allows a search step to a nonpermissible neighbour if it leads to an improvement in the incumbent solution.

The performance of simple TS algorithms can be very impressive, but usually depends strongly on the value of the tabu tenure parameter. Several extensions have been proposed that adjust the tabu tenure while running the search process; the best known example is reactive TS [29] (see also Chapter 21). Furthermore, additional intermediate and long-term memory mechanisms can be used to further improve the effectiveness of simple TS. These mechanisms aim to either intensify the search in particular regions of the search space or to diversify the search to prevent stagnation. For a detailed description of these techniques, we refer to the book by Glover and Laguna [28]. More detailed information on TS can also be found in Chapter 23 of this book.

Dynamic Local Search

A different approach for escaping from local optima provides the basis for DLS: Rather than explicitly using worsening search steps, DLS modifies the given evaluation function whenever its subsidiary local search procedure encounters a local optimum. More precisely, DLS works as follows. Initially, a subsidiary local search algorithm, typically an iterative improvement procedure, is executed until a local optimum s is encountered. At this point, the evaluation function is modified such that s is no longer a local optimum. Then, the subsidiary local search is run again until it reaches a local optimum of the modified evaluation function, at which point, further modifications of the evaluation function are performed. These phases of local search followed by evaluation function modifications are repeated until a termination criterion is satisfied.

The modifications to the evaluation function are typically achieved by means of *penalty weights* that are associated with individual solution components. The augmented evaluation function effectively used by the subsidiary local search algorithm is then of the form

$$g_p(s) := g(s) + \sum_{i \in C(s)} penalty(i) \tag{19.2}$$

where $g(s)$ is the original evaluation function, $C(s)$ the set of solution components of candidate solution s, and $penalty(i)$ the penalty of solution component i of s. At the beginning of the search process, all penalties are set to zero. DLS algorithms mainly differ in the way the penalty update is performed, for example, the usage of additive versus multiplicative increments or the selection of the solution components whose penalties are increased. Typically, when a local optimum s is encountered, penalties are only increased for solution components occurring in s. For example, in guided local search (GLS), a well-known family of DLS algorithms [30,31], the following penalty update mechanism is used. When encountering a local minimum s of g_p, GLS computes a utility value $u(i) := g_i(s)/(1+penalty(i))$ for each solution component i of s, where $g_i(s)$ is the contribution of i to the original evaluation (or objective) function. (For example, in the case of the TSP, the solution components are typically edges of the given graph, and $g_i(s)$ is the cost of edge i.) Then, only the penalties of solution components with maximal utility are increased. Note that the term $1 + penalty(i)$ in the denominator of the definition of $u(i)$ avoids overly frequent penalisation of solution components.

In addition to penalty increases in local optima, many DLS algorithms also occasionally decrease penalty values. DLS algorithms have been demonstrated to achieve state-of-the-art performance for various combinatorial problems, including SAT [32]. They are conceptually related to Lagrangian methods in continuous optimisation.

19.4 Hybrid SLS Methods

While the previously discussed SLS methods provide the basis for state-of-the-art algorithms for many hard combinatorial problems, often, further performance improvements can be achieved by combining various SLS strategies into hybrid algorithms. In fact, RII can already be seen as a hybrid SLS method, since it combines two different types of search steps—random walk and iterative improvement steps. In the following, we briefly discuss several other hybrid SLS methods. In many cases, hybrid SLS methods combine constructive search steps and perturbative local search steps, or larger modifications of candidate solutions with the application of "simple" local search algorithms.

Iterated Local Search

The key idea behind this SLS method is to escape from local optima of the given evaluation function by means of a perturbation mechanism. Essentially, three components form the core of any ILS algorithm. A subsidiary local search procedure is used to efficiently find local optima; this is typically based on an iterative improvement algorithm or a "simple" SLS method. The perturbation procedure introduces a modification to a given candidate solution to allow the search process to escape from local optima. Finally, an acceptance criterion is used to decide whether the search should be continued from a newly found local optimum. Based on these components, ILS works as outlined in Figure 19.3.

```
Iterated Local Search (ILS):
  determine initial candidate solution $s$
  perform subsidiary local search on $s$
  While termination criterion is not satisfied:
    $r := s$
    perform perturbation on $s$
    perform subsidiary local search on $s$
    based on acceptance criterion, keep $s$ or revert to $s := r$
```

FIGURE 19.3 Outline of iterated local search.

When using an iterative improvement algorithm as the subsidiary local search procedure, ILS can be seen as performing a biased random walk in the space of local optima reached by this procedure. The perturbation procedure should introduce a modification that cannot be immediately reversed by the local search procedure, to achieve effective diversification of the search. The acceptance criterion determines the degree of search intensification; for example, if only improving candidate solutions are accepted, ILS performs randomised first-improvement search in the space of local optima. Together, the perturbation procedure and the acceptance criterion determine the balance between search intensification and diversification.

One of the major attractions of ILS stems from the fact that it is typically very easy to extend an existing implementation of a simple local search algorithm into a basic ILS algorithm. ILS algorithms define the current state-of-the-art for solving many hard combinatorial problems, the most prominent of which is the TSP [33]. Conceptually, there is a close relationship between ILS and some advanced forms of TS; furthermore, the previously mentioned basic and skewed variable neighbourhood search algorithms can be seen as special cases of ILS. For more details on ILS we refer to Lourenço et al. [34].

Iterated Greedy Algorithms

Greedy construction methods are at the core of many well-known approximation algorithms. They also provide the basis for iterated greedy (IG) algorithms, which can be seen as a variant of ILS in which local search and perturbation phases are replaced by construction and destruction phases. Starting from a complete candidate solution, IG alternates between phases of destructive and constructive search. During a destruction phase, some solution components are removed from the current candidate solution s (e.g., uniformly at random or heuristically, depending on their impact on the evaluation function), resulting in some partial (nonempty) candidate solution s_p. During a construction phase, solution components are added heuristically, starting from s_p, until a new complete candidate solution s' has been obtained. As in ILS, an acceptance criterion is used to decide whether to continue the search from s' or s.

Note that the initial candidate solution in IG may be generated by a construction method that may be different from the one used in subsequent construction phases. Also, a perturbative local search procedure may be used to further improve any complete candidate solutions considered during the search. In this case, IG may be seen as a variant of ILS in which a combination of a destruction and a construction phase forms the perturbation procedure. IG methods are a promising approach for solving problems for which good constructive algorithms exist. Problems for which IG algorithms reach state-of-the-art performance include set covering [35,36] and flow-shop scheduling [37].

Greedy Randomised Adaptive Search Procedures

A combination of constructive and perturbative local search forms the basis for GRASP [2,3], which works as follows: Using a randomised construction procedure, a complete candidate solution is generated, which is then improved using a perturbative local search procedure; this two-phase process is repeated until a termination criterion is satisfied.

The construction procedure in GRASP iteratively adds solution components that are chosen randomly from a *restricted candidate list*; the elements of this list are determined using a heuristic function, whose value for a given solution component may depend on the solution components already present in the current partial candidate solution (this is the “adaptive” aspect of the search process). GRASP has been applied to a broad range of combinatorial problems; for a detailed description on GRASP and various extensions we refer to Resende and Ribeiro [3].

19.5 Population-Based SLS Methods

Various SLS methods maintain a population of candidate solutions which are simultaneously manipulated in each search step. As previously noted, such population-based methods can be formulated within the unified SLS framework described in Section 19.1 by considering search positions that consist of sets of candidate solutions in combination with suitably defined neighbourhood relations as well as initialisation and step functions. The appeal of population-based SLS methods partly stems from the fact that the use of a population provides a straightforward means for achieving search diversification; however, compared with other SLS methods, this often comes at the cost of higher implementation effort and more parameters that need to be tuned to achieve competitive performance. The two widely known population-based methods described in the following are inspired by biological mechanisms; however, it should be noted that there are many other population-based SLS methods, such as population-based extensions of ILS, that are not motivated in this way.

Ant Colony Optimisation

The inspiration for ACO stems from the pheromone trail laying and following behaviour exhibited by several ant species [5]. In ACO, the (artificial) ants are randomised construction procedures that take into account (artificial) pheromone trails and heuristic information when iteratively constructing complete candidate solutions. Essentially, the pheromone trails are modelled by numerical values that are associated with solution components; these are adapted while solving a given problem instance and hence reflect the search experience of the (artificial) ant colony.

In each construction step, solution components are chosen with a probability that is proportional to their pheromone value and the heuristic information. For example, in the TSP, the first problem tackled by ACO algorithms [4], a pheromone value τ_{ij} is associated with each edge (i, j) of the given graph, and the heuristic information η_{ij} is typically defined as the inverse of the length of edge (i, j). Hence, an ant located at a vertex i would add vertex $j \in N(i)$ to its current partial tour s_p with probability

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in N(i)} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}}, \tag{19.3}$$

where $N(i)$ is the feasible neighbourhood of vertex i, i.e., the set of all vertices that have not yet been visited in s_p, and α and β are parameters that control the relative influence of pheromone values and heuristic information, respectively. This probabilistic selection mechanism was used in Ant System, the first ACO algorithm [4,38]; the tour construction procedure applied by the artificial ants using this mechanism resembles a probabilistic variant of the nearest neighbour construction heuristic.

After all ants have constructed a complete candidate solution, many ACO algorithms apply a perturbative local search procedure (such as iterative improvement) to each complete candidate solution; it has been shown that in the context of solving $\mathcal{NP}$-hard problems, the overall performance of an ACO algorithm often depends crucially on this local search phase [5,39]. Then, the pheromone values are updated in two steps. In the first step (which models pheromone evaporation), all pheromone values are decreased, typically by multiplication with a constant factor. In the second step (which models pheromone deposit), the pheromone values of the solution components contained in one or more of the current candidate solutions are increased; the amount of increase often depends on the quality of the respective solutions. Cycles of construction (and subsidiary local search) phases followed by pheromone updates are repeated until a termination criterion is satisfied.

Several variants of ACO have been proposed that all share the same basic underlying ideas (see the book by Dorigo and Stützle [5] for an overview). The ACO metaheuristic [40] gives a general framework for these variants. In general, ACO is steadily gaining popularity and provides the basis for state-of-the-art algorithms for a number of hard combinatorial optimisation problems [5]. More information on ACO can be found in Chapter 26 of this book.

Evolutionary Algorithms

One of the most prominent classes of population-based SLS methods has been inspired by concepts from biological evolution. Evolutionary algorithms start with an initial set of candidate solutions (which can be generated randomly or by means of heuristic construction search methods), and iteratively modify this population by means of three types of operations: mutation, recombination, and selection.

Typical *mutation operators* introduce small perturbations to candidate solutions in a randomised fashion; the amount of perturbation applied is usually controlled by a parameter called the *mutation rate*.

Recombination operators generate one or more new candidate solutions, often called “offspring”, by combining information from two or more “parent” candidate solutions. One of the most common types of recombination is known as *crossover*; it generates offspring by assembling partial candidate solutions from two parents. *Selection operators* are used to determine which candidate solutions from the current population and from the set of new candidate solutions obtained from mutation and recombination will form the population used in the next iteration of the search process. This choice is typically based on the value of candidate solutions under the given evaluation function (which, in the context of EAs, is commonly referred to as *fitness function*), such that better candidate solutions have a higher probability to “survive” the selection process. Selection operators are also used for choosing candidate solutions from the current population to undergo mutation or recombination.

The performance of EAs depends crucially on the choice of the evolutionary operators. In general, the performance of EAs improves if knowledge about the given problem is exploited in the operators that are applied. In fact, much research in EAs has been devoted to the design of effective mutation and crossover operators; a good example for this is the TSP [41,42]. As in the case of ACO, substantial performance improvements can often be achieved by additionally optimising candidate solutions using a perturbative local search method, such as iterative improvement. The resulting hybrid algorithms are also known as memetic algorithms (MAs) [43]. For a detailed account on MAs and their performance we refer to Chapter 27 of this book. *Scatter search* and *path relinking* are SLS methods whose roots can be traced back to the mid-1970s [44] and that only recently have regained considerable attention. They can be seen as MAs that use special types of recombination and selection operators. For details on these methods, we refer to the work of Glover et al. [45] and Laguna and Martí [46].

19.6 Discussion and Research Directions

The study of SLS methods lies at the intersection of computing science, operations research, statistics, and various application areas. Their successful application requires knowledge of the various SLS techniques and their properties; despite the generic nature of many SLS methods, insights into the problem to be solved are often also crucial.

Overall, the behaviour of most high-performance SLS algorithms is not well understood, and although some theoretical results exist (e.g., for SA and ACO algorithms), these are often obtained under specific assumptions that limit their practical relevance [47,48]. Still, some theoretical properties, such as empirical approximate completeness (which guarantees that when run arbitrarily long, an algorithm finds a solution to any given, soluble problem instance with probability approaching one), have been shown to be useful in guiding the development of high-performance SLS algorithms (see, e.g., Ref. [49]).

Nevertheless, the analysis of SLS algorithms relies mostly on computational experiments, and advanced methods for the empirical analysis of SLS behaviour play an important role in the development of new algorithms. Empirical studies in this area are complicated by the fact that most SLS algorithms are heavily randomised; but using advanced empirical methods, such as the approach based on run-time distributions discussed in Chapter 14, have contributed significantly to an improved understanding of SLS behaviour as well as to the development of high-performance SLS algorithms for many hard combinatorial problems [49,50]. Yet, compared with mature empirical sciences, such as physics or biology, computing science in general, and the area of algorithmics in particular, are still in the early stages of adopting and exploiting a well-founded empirical approach.

In the past, much of the successful work on SLS algorithms and generic SLS methods has relied crucially on prior experience and good intuitions. One of the major directions in SLS research concerns the development of improved practices for developing SLS algorithms and for their application to new problems. In this context, it is likely that conceptual frameworks for the design and exploration of SLS methods, as well as implementation and experimentation environments will play an important role [51,52]. Furthermore, an improved understanding of the relationship between the features of problems and problem instances on one hand, and properties and behaviour of SLS methods on the other will likely provide the basis for more principled approaches for the design and application of SLS algorithms. Insights to be gained in this context are also of considerable scientific interest. Progress in this research direction is significantly leveraged by advanced search space analysis techniques, statistical methods, and machine learning approaches (see, e.g., Merz and Freisleben [53] or Watson et al. [54]).

The formalisation of general principles underlying the design of successful SLS algorithms remains a major challenge. One interesting question in this context concerns the role and degree of randomisation. While deterministic local search algorithms exist, most high-performance SLS algorithms are highly randomised. It is not clear to which extent it is generally possible to derandomise these algorithms without major losses in robustness or peak performance. Interestingly, there is evidence that in most cases, the quality of the random number source used by the algorithm is not critical and that the number of randomised decisions can often be significantly reduced [55]. However, in most application areas, the use of deterministic algorithms is not inherently preferable, and randomised algorithms have some general advantages, for example, with respect to straightforward and efficient parallelisation techniques. Therefore, while from a scientific point of view, the derandomisation of SLS algorithms poses interesting questions and challenges, its practical importance is somewhat unclear.

Finally, many challenges remain in the context of SLS methods to more complex combinatorial problems, including multiobjective, dynamic, and stochastic problems. Furthermore, there appears to be considerable potential in the context of SLS methods for solving continuous optimisation problems and hybrid problems with discrete and continuous components. Overall, there is no doubt that as one of the most versatile and successful approaches for solving hard combinatorial problems, SLS methods will continue to attract the attention of researchers and practitioners from many academic and application areas.

Acknowledgments

Holger H. Hoos acknowledges support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Discovery Grant 238788-05; and Thomas Stützle acknowledges support of the Belgian FNRS, of which he is a research associate.

References

[1] Hoos, H. H. and Stützle, T., *Stochastic Local Search—Foundations and Applications*, Morgan Kaufmann Publishers, San Francisco, CA, 2004.

[2] Feo, T. A. and Resende, M. G. C., A probabilistic heuristic for a computationally difficult set covering problem, *Oper. Res. Lett.*, 8(2), 67, 1989.

[3] Resende, M. G. C. and Ribeiro, C. C., Greedy randomized adaptive search procedures, in *Handbook of Metaheuristics*, Glover, F. and Kochenberger, G., Eds., Kluwer Academic Publishers, Norwell, 2002, p. 219.

[4] Dorigo, M., Maniezzo, V., and Colorni, A., Positive Feedback as a Search Strategy, 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.

[5] Dorigo, M. and Stützle, T., *Ant Colony Optimization*, MIT Press, Cambridge, MA, 2004.

[6] Voß, S., Martello, S., Osman, I. H., and Roucairol, C., Eds., *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, Dordrecht, 1999.

[7] Bentley, J. L., Fast algorithms for geometric traveling salesman problems, *ORSA J. Comput.*, 4(4), 387, 1992.

[8] Jain, A. S., Rangaswamy, B., and Meeran, S., New and "stronger" job-shop neighbourhoods: a focus on the method of Nowicki and Smutnicki, *J. Heuristics*, 6(4), 457, 2000.

[9] Yannakakis, M., The analysis of local search problems and their heuristics, *Proc. of STACS*, Lecture Notes in Computer Science, Vol. 415, Springer, Berlin, 1990, p. 298.

[10] Martin, O. C., Otto, S. W., and Felten, E. W., Large-step Markov chains for the traveling salesman problem, *Complex Syst.*, 5(3), 299, 1991.

[11] Hansen, P. and Mladenović, N., Variable neighborhood search: principles and applications, *Eur. J. Oper. Res.*, 130(3), 449, 2001.

[12] Hansen, P. and Mladenović, N., Variable neighborhood search, in *Handbook of Metaheuristics*, Glover, F. and Kochenberger, G., Eds., Kluwer Academic Publishers, Norwell, 2002, p. 145.

[13] Ahuja, R. K., Ergun, O., Orlin, J. B., and Punnen, A. P., A survey of very large-scale neighborhood search techniques, *Disc. Appl. Math.*, 123(1–3), 75, 2002.

[14] Kernighan, B. W. and Lin, S., An efficient heuristic procedure for partitioning graphs, *Bell Syst. Tech. J.*, 49, 213, 1970.

[15] Lin, S. and Kernighan, B. W., An effective heuristic algorithm for the traveling salesman problem, *Oper. Res.*, 21(2), 498, 1973.

[16] Glover, F., Ejection chain, reference structures and alternating path methods for traveling salesman problems, *Disc. Appl. Math.*, 65(1–3), 223, 1996.

[17] Thompson, P. M. and Orlin, J. B., The Theory of Cycle Transfers, Working paper OR 200-89, Operations Research Center, MIT, Cambridge, MA, 1989.

[18] Thompson, P. M. and Psaraftis, H. N., Cyclic transfer algorithm for multivehicle routing and scheduling problems, *Oper. Res.*, 41, 935, 1993.

[19] Ahuja, R. K., Orlin, J. B., and Sharma, D., Multi-Exchange Neighbourhood Structures for the Capacitated Minimum Spaning Tree Problem, Working paper, 2000.

[20] Potts, C. N. and van de Velde, S., Dynasearch: Iterative Local Improvement by Dynamic Programming; Part I, the Traveling Salesman Problem, TR LPOM–9511, Faculty of Mechanical Engineering, University of Twente, Enschede, The Netherlands, 1995.

[21] Congram, R. K., Potts, C. N., and van de Velde, S., An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, *INFORMS J. Comput.*, 14(1), 52, 2002.

[22] Selman, B., Kautz, H., and Cohen, B., Noise strategies for improving local search, *Proc. National Conf. on Artificial Intelligence*, AAAI Press/The MIT Press, Cambridge, MA, 1994, p. 337.

[23] Connolly, D. T., An improved annealing scheme for the QAP, *Eur. J. Oper. Res.*, 46(1), 93, 1990.

[24] Fielding, M., Simulated annealing with an optimal fixed temperature, *SIAM J. Optimization*, 11(2), 289, 2000.

[25] Kirkpatrick, S., Gelatt, C. D., Jr., and Vecchi, M. P., Optimization by simulated annealing, *Science*, 220, 671, 1983.

[26] Cerný, V., A thermodynamical approach to the traveling salesman problem, *J. Optimization Theor. Appl.*, 45(1), 41, 1985.

[27] Dueck, G. and Scheuer, T., Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing, *J. Comput. Phy.*, 90(1), 161, 1990.

[28] Glover, F. and Laguna, M., *Tabu Search*, Kluwer Academic Publishers, Boston, 1997.

[29] Battiti, R. and Tecchiolli, G., Simulated annealing and tabu search in the long run: a comparison on QAP tasks, *Comput. Math. Appl.*, 28(6), 1, 1994.

[30] Voudouris, C., Guided Local Search for Combinatorial Optimization Problems, Ph.D. thesis, University of Essex, Department of Computer Science, Colchester, UK, 1997.

[31] Voudouris, C. and Tsang, E., Guided local search and its application to the travelling salesman problem, *Eur. J. Oper. Res.*, 113(2), 469, 1999.

[32] Hutter, F., Tompkins, D. A. D., and Hoos, H. H., Scaling and probabilistic smoothing: efficient dynamic local search for SAT, in *Principles and Practice of Constraint Programming*, Van Hentenryck, P., Ed., Lecture Notes in Computer Science, Vol. 2470, Springer, Berlin, 2002, p. 233.

[33] Johnson, D. S. and McGeoch, L. A., Experimental analysis of heuristics for the STSP, in *The Traveling Salesman Problem and its Variations*, Gutin, G. and Punnen, A., Eds., Kluwer Academic Publishers, Dordrecht, 2002, p. 369.

[34] Lourenço, H. R., Martin, O., and Stützle, T., Iterated local search, in *Handbook of Metaheuristics*, Glover, F. and Kochenberger, G., Eds., Kluwer Academic Publishers, Norwell, 2002, p. 321.

[35] Jacobs, L. W. and Brusco, M. J., A local search heuristic for large set-covering problems, *Naval Res. Logistics Quart.*, 42(7), 1129, 1995.

[36] Marchiori, E. and Steenbeek, A., An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling, in *Real-World Applications of Evolutionary Computing*, Cagnoni, S., et al., Eds., Lecture Notes in Computer Science, Vol. 1803, Springer, Berlin, 2000, p. 367.

[37] Ruiz, R. and Stützle, T., A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, *Eur. J. Oper. Res.*, 177(3), 2033, 2007.

[38] Dorigo, M., Maniezzo, V., and Colorni, A., Ant System: optimization by a colony of cooperating agents, *IEEE Trans. Syst., Man, Cybern.—Part B*, 26(1), 29, 1996.

[39] Stützle, T. and Hoos, H. H., $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System, *Future Generation Comput. Syst.*, 16(8), 889, 2000.

[40] Dorigo, M. and Di Caro, G., The ant colony optimization meta-heuristic, in *New Ideas in Optimization*, Corne, D., Dorigo, M., and Glover, F., Eds., McGraw Hill, London, 1999, p. 11.

[41] Potvin, J. Y., Genetic algorithms for the traveling salesman problem, *Ann. Oper. Res.*, 63, 339, 1996.

[42] Merz, P. and Freisleben, B., Memetic algorithms for the traveling salesman problem, *Complex Syst.*, 13(4), 297, 2001.

[43] Moscato, P., Memetic algorithms: a short introduction, in *New Ideas in Optimization*, Corne, D., Dorigo, M., and Glover, F., Eds., McGraw Hill, London, 1999, p. 219.

[44] Glover, F., Heuristics for integer programming using surrogate constraints, *Decision Sci.*, 8, 156, 1977.

[45] Glover, F., Laguna, M., and Martí, R., Scatter search and path relinking: advances and applications, in *Handbook of Metaheuristics*, Glover, F. and Kochenberger, G., Eds., Kluwer Academic Publishers, Norwell, 2002, p. 1.

[46] Laguna, M. and Martí, R., *Scatter Search: Methodology and Implementations in C*, Vol. 24, Kluwer Academic Publishers, Dordrecht, 2003.

[47] Hajek, B., Cooling schedules for optimal annealing, *Math. Oper. Res.*, 13(2), 311, 1988.

[48] Gutjahr, W. J., ACO algorithms with guaranteed convergence to the optimal solution, *Inf. Proc. Lett.*, 82(3), 145, 2002.

[49] Hoos, H. H., On the run-time behaviour of stochastic local search algorithms for SAT, *Proc. National Conf. on Artificial Intelligence*, AAAI Press/The MIT Press, Cambridge, MA, 1999, p. 661.

[50] Stützle, T. and Hoos, H. H., Analysing the run-time behaviour of iterated local search for the travelling salesman problem, in *Essays and Surveys on Metaheuristics*, Hansen, P. and Ribeiro, C. C., Eds., Kluwer Academic Publishers, Norwell, 2001, p. 589.

[51] Van Hentenryck, P. and Michel, L., *Constraint-Based Local Search*, MIT Press, Cambridge, MA, 2005.

[52] Voß, S. and Woodruff, D. L., Eds., *Optimization Software Class Libraries*, Vol. 18, Kluwer Academic Publishers, Dordrecht, 2002.

[53] Merz, P. and Freisleben, B., Fitness landscapes and memetic algorithm design, in *New Ideas in Optimization*, Corne, D., Dorigo, M., and Glover, F., Eds., McGraw Hill, London, 1999, p. 244.

[54] Watson, J.-P., Whitley, L. D., and Howe, A. E., Linking search space structure, run-time dynamics, and problem difficulty: a step towards demystifying tabu search, *J. Artif. Intell. Res.*, 24, 221, 2005.

[55] Hoos, H. H. and Tompkins, D. A. D., On the quality and quantity of random decisions in stochastic local search for SAT, in *Advances in Artificial Intelligence, Conf. of the Canadian Society for Computational Studies of Intelligence,* Lamontagne, L. and Marchand, M., Eds., Vol. 4013, Springer Verlag, 2006, pp. 146–158.

20

Very Large-Scale Neighborhood Search: Theory, Algorithms, and Applications

Ravindra K. Ahuja
University of Florida

Özlem Ergun
Georgia Institute of Technology

James B. Orlin
Massachusetts Institute of Technology

Abraham P. Punnen
Simon Fraser University

20.1 Introduction

A combinatorial optimization problem (COP) P consists of a collection of instances. An instance of P can be represented as an ordered pair (F, f), where F is the family of feasible solutions and f the objective function, which is used to compare two feasible solutions. The family F is formed by subsets of a finite set $\mathsf{E} = \{1, 2, \ldots, m\}$ called the ground set. The objective function $f : \mathsf{F} \to Q^+ \cup \{0\}$ assigns a nonnegative cost to every feasible solution S in F. The Traveling Salesman Problem (TSP) and the Minimum Spanning Tree Problem are typical examples of COPs.

Most of the COPs of practical interest are NP-hard. Local search is one of the primary solution approaches for computing an approximate solution for such hard problems. To describe a local search algorithm formally, we need the concept of a neighborhood for each feasible solution. A *neighborhood function* for an instance (F, f) of a COP P is a mapping $N_{\mathsf{F}} : \mathsf{F} \to 2^{\mathsf{F}}$. We assume that N_{F} does not depend on the objective function f. For convenience, we usually drop the subscript and simply write N. For a feasible solution S, $N(S)$ is called the neighborhood of S. We assume that$S \in N(S)$. For a minimization problem, a feasible solution $\overline{S} \in \mathsf{F}$ is said to be *locally optimal* with respect to N if $f(\overline{S}) \leq f(S)$ for all$S \in N(\overline{S})$. Then the *local search problem* is to find a locally optimal solution for a given COP.

Classic neighborhood functions studied in the combinatorial optimization literature include the 2- and 3-opt neighborhoods for the TSP [1,2], the flip neighborhood for Max Cut [3], and the swap neighborhood for Graph Partitioning [3,4]. The sizes of these neighborhoods are polynomial in the problem size. However, the class of local search algorithms that we are considering here primarily concentrates on neighborhoods of very large size, often exponential in the problem size. For the TSP, this class includes variable depth neighborhoods, ejection chain neighborhoods, pyramidal tours neighborhoods, permutation tree neighborhoods, neighborhoods induced by polynomial-time solvable special cases, etc. For partitioning problems, various multiexchange neighborhoods studied in literature have this property.

Roughly speaking, a local search algorithm starts with an initial feasible solution and then repeatedly searches the neighborhood of the "current" solution to find better solutions until it reaches a locally optimal solution. Computational studies of local search algorithms and their variations have been extensively reported in the literature for various COPs (see, e.g., Refs. [2,5] for studies of the Graph Partitioning Problem and the TSP, respectively). Empirically, local search heuristics appear to converge rather quickly, within low-order polynomial time. In general, the upper bound on the guaranteed number of improving moves is pseudopolynomial.

This chapter is organized as follows. In Section 20.2, we consider various applications of very large scale neighborhood (VLSN) search algorithms and discuss in detail how to develop such algorithms using multiexchanges for various partitioning problems. Section 20.3 deals with theoretical concepts of extended neighborhoods and linkages with domination analysis of algorithms. The results in this section reaffirm the importance of using large-scale neighborhoods in local search. In Section 20.4, we deal with performance guarantees for computing an approximation to a local minimum. This is especially relevant for VLSN search. Searching a large-scale neighborhood is sometimes NP-hard and thus approximation algorithms are used. In these cases, the algorithm may terminate at a point that is not a local minimum.

20.2 VLSN Search Applications

VLSN search algorithms have been successfully applied to solve various optimization problems of practical interest [53–62]. This includes the capacitated minimum spanning tree (CMST) problem [6–8], vehicle routing problems [9–12], the TSP [11,13–17], the weapon target assignment problem [18], the generalized assignment problem [19–21], the plant location problem [22], parallel machine scheduling problems [23], airline fleet assignment problems [24–26], the quadratic assignment problem [27], pickup and delivery problems with time windows [4,28], the multiple knapsack problem [29], manufacturing problems [30], optimization problems in IMRT treatment planning [31], school timetabling problems [32], the graph coloring problem [33], etc. The successful design of an effective VLSN search algorithm depends on the ability to identify a good neighborhood function and the ability to design an effective exact or heuristic algorithm to search the neighborhood for an improved solution. A VLSN search algorithm can be embedded within a metaheuristic framework, such as tabu search [34], genetic algorithms [35], scatter search [36], and GRASP [37] to achieve further enhances in performance. Simulated annealing may be used in principle, but is not likely to be a successful approach when the size of the neighborhood is exponentially large.

Researchers have used various techniques for developing good neighborhood functions that lead to effective VLSN search algorithms. This includes multiexchange [6,7,38,39], ejection chains [40], variable depth methods [41,42], integer programming [10], weighted matching [10,13,14,16,17], set partitioning [10], etc. A comprehensive discussion of all these techniques is outside the scope of this chapter. Applications of ejection chains in local search are discussed in other chapters of this book and hence we will not discuss it here. To illustrate the features of a VLSN search algorithm, we primarily concentrate on applications of multiexchange neighborhoods originally developed by Thompson and Orlin [43] and Thompson and Psaraftis [38]. It is one of the successful approaches for developing VLSN search algorithms for various partitioning problems [6,7,29].

20.2.1 Partitioning Problems and Multiexchange Neighborhoods

A subset B of the ground set E is said to be a *feasible subset* if it satisfies a collection of prescribed conditions called *feasibility conditions*. We assume there is a *feasibility oracle* ζ that verifies whether a subset B of E satisfies the feasibility conditions. A partition $S = \{S_1, S_2, \dots, S_p\}$ is said to be *feasible* if S_i is a feasible subset for each $i = 1, 2, \dots, p$. Let $c : 2^{\mathsf{E}} \to \mathbb{Q}$ be a prescribed cost function. The cost $C(S)$ of the

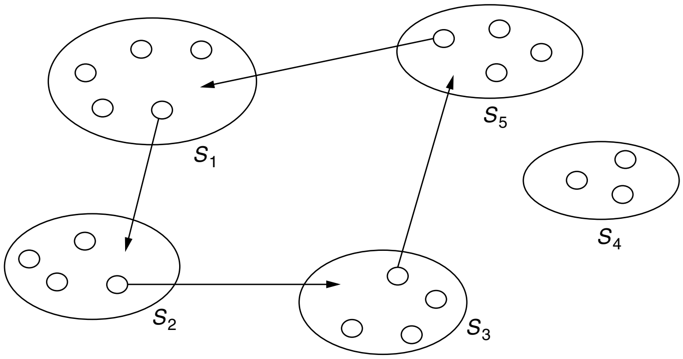


FIGURE 20.1 A cyclic exchange.

partition $S = \{S_1, S_2, \ldots, S_p\}$ is defined as $C(S) = \sum_{i=1}^{p} c(S_i)$. *Partitioning problems* are the subclass of COP, where the family of feasible solutions F is the collection of all feasible partitions of E with $C(.)$ as the objective function.

Partitioning problems are typically NP-hard even if the feasibility oracle ζ and evaluation of the cost $c(.)$ of the subsets of a partition run in polynomial time. Several well-studied COPs are special cases (or may be viewed as special cases) of partitioning problems. These include the CMST problem [6,7,44], the generalized assignment problem [19–21], vehicle routing problems [10,23], graph partitioning problems [4,42], the k-constrained multiple knapsack problem [29], etc. We will discuss some of these problems in detail later. Next, we develop a VLSN search algorithm for the general partitioning problem.

The first step in the design of a VLSN search algorithm is to develop a suitable neighborhood function.

Let $S = \{S_1, S_2, \ldots, S_p\}$ be a feasible partition. A *cyclic exchange* of S selects a collection $\{S_{\pi_1}, S_{\pi_2}, \ldots, S_{\pi_q}\}$ of subsets from S and moves an element say b_i from subset S_{π_i} to subset $S_{\pi_{i+1}}$ for i =1, 2, …, q, where $q + 1 = 1$ (see Figure 20.1). Let S^* be the resulting partition. If S^* is a feasible partition, then we call the exchange a *feasible cyclic exchange.* It can be verified that

$$C(S^*) = C(S) + \sum_{i=1}^{q} (c(\{b_{i-1}\} \cup S_{\pi_i} \backslash \{b_i\}) - c(S_{\pi_i})) \tag{20.1}$$

where b_0 is defined to be b_q. If $q = 2$, the cyclic exchange reduces to a 2-exchange, an operation well studied for various partitioning problems [5,15,45]. In a (feasible) cyclic exchange, if we omit the move from S_{π_q} to $S_{\pi_{q+1}} (= S_{\pi_1})$, the resulting exchange is called *a (feasible)path exchange.* For $q = 2$, a path exchange reduces to the shift operation, again well studied for various partitioning problems [45].

The cyclic and path exchanges discussed above were introduced by Thompson and Orlin [43] and Thompson and Psaraftis [38] and subsequently used by several researchers in the context of various partitioning problems.

Given a feasible partition S, let Z(S) be the collection of all feasible partitions of E that can be obtained by cyclic exchanges from S. Similarly, let P(S) be the collection of feasible partitions of E that can be obtained by a path exchange from S. We call Z(S) the *cyclic exchange neighborhood* of S, and we call P(S) the *path exchange neighborhood* of S. The union Z(S)∪P(S) is called the *multiexchange neighborhood* of S. The cardinality of the neighborhoods Z(S) and P(S) is often exponential. A precise estimate of this cardinality depends on various parameters including the nature of the feasibility oracle ζ. Thus Z(S) and P(S) qualify as very large-scale neighborhoods that can be used in the design of a VLSN search algorithm.

Besides the size of the neighborhood, the power of a VLSN search algorithm also depends on our ability to obtain an improved solution in the neighborhood. Given a feasible solution S, the problem of finding an improved solution in a given neighborhood is called a *local augmentation problem* [46]. We next discuss how to solve the local augmentation problem for Z(S). This is achieved by (approximately) solving an optimization problem on an associated structure called the *improvement graph.*

The improvement graph associated with a feasible partition is denoted by $G(S)$. Its node set is E, which is also the ground set. Given a partition S, for each element i of E let $\theta(i)$ denote the index of the subset in which it belongs in the partition S. That is, for $i \in S_r$, $\theta(i) = r$. The arc set $A = A(G(S))$ of $G(S)$ is constructed as follows. An arc $(i, j) \in A$ signifies that element i leaves subset $S_{\theta(i)}$ and enters subset $S_{\theta(j)}$ while element j leaves $S_{\theta(j)}$. So,

$$A = \left\{(i, j) : \ i, j \in E, \ \theta(i) \neq \theta(j) \ \text{ and } \ \{i\} \cup S_{\theta(j)} \backslash \{j\} \text{ is a feasible subset}\right\}$$

The cost α_{ij} of arc $(i, j) \in A$ is given by

$$\alpha_{ij} = c(\{i\} \cup S_{\theta(j)} \backslash \{j\}) - c(S_{\theta(j)}) \tag{20.2}$$

A directed cycle $\langle i_1 - i_2 - \cdots - i_r - i_1 \rangle$ in the improvement graph is said to be *subset disjoint* if $\theta(i_p) \neq \theta(i_q)$ for $p \neq q$. In this definition, if we replace "cycle" by "path" we get a *subset-disjoint directed path.*

Lemma 20.1 **(Thompson and Orlin [43])**

There is a one-to-one cost-preserving correspondence between cyclic exchanges with respect to S and subset-disjoint directed cycles in G (S).

A negative cost subset-disjoint directed path (cycle) in $G(S)$ is called a *valid path* (*cycle*).

In view of Lemma 20.1, a valid cycle in $G(S)$ yields an improved partition. If $G(S)$ contains no valid cycle, then S is locally optimal with respect to $\mathsf{N}(S)$. Unfortunately, finding such a valid cycle is NP-hard [43]. Thus, the local augmentation problem for the neighborhood $\mathsf{N}(S)$ is also NP-hard.

20.2.1.1 Local Augmentation Algorithm for N(S)

Ahuja et al. [7] proposed a heuristic algorithm to compute a valid cycle by modifying the label correcting algorithm for shortest paths. An exact algorithm based on dynamic programming (implicit enumeration) to solve this problem was introduced in Ref. [6]. We briefly discuss the algorithm, and refer the reader to Ref. [6] for further details.

For any subgraph H of $G(S)$, its cost $\alpha(S)$ is given by $\alpha(S) = \sum_{ij \in P} \alpha_{ij}$, where *ij* is shorthand for (i, j). For a directed path P in $G(S)$, let *tail*(P) denote the start node, *head*(P) denote the end node, and *label*(P) denote the set $\{\theta(i) \ : \ i \in P\}$. We say that a path P_1 dominates another path P_2 in $G(S)$ if $\alpha(P_1) < \alpha(P_2)$, *tail*($P_1$)= *tail*($P_2$), *head*($P_1$)= *head*($P_2$), and *label*($P_1$)= *label*($P_2$). If P_1 dominates P_2 and P_2 is part of a valid cycle $\mathbb{C}$, then clearly $P_1 \cup \mathbb{C} \backslash P_2$ contains a valid cycle with cost no more than $\alpha(\mathbb{C})$. For each path P, we associate a triplet (*tail*(P), *head*(P), *label*(P)) called the *key value* of P. In our search for a valid cycle in $G(S)$, among paths with the same key value, we only need to consider one with the smallest cost. The following lemma further cuts down the number of paths needed to be considered.

Lemma 20.2 **(Lin and Kernighan [41])**

If $W = \langle i_1 - i_2 - \cdots - i_r - i_1 \rangle$ is a negative cost cycle, then there exists a node i_h in W such that each directed path $i_h - i_{h+1}$, $i_h - i_{h+1} - i_{h+2}, \ldots, i_h - i_{h+1} - i_{h+2} - \ \ldots \ - i_{h-1}$ (where indexes are modulo r) is a negative cost-directed path.

In view of Lemma 20.2 and the preceding discussions, we need to only consider nondominated valid paths as candidates for forming valid cycles. Let Π_k be the set of all valid nondominated paths of length k in $G(S)$. The algorithm of Ahuja et al. [6] progressively generates Π_k for larger values of k. The algorithm enumerates valid paths using a forward dynamic programming recursion. It first obtains all valid paths of length 1 and uses these paths to generate valid paths of length 2, etc. This process is repeated until we have obtained all valid paths of length R for some given length R, or until we find a valid cycle. From each valid path, candidate cycles are examined by connecting the head node with the tail node by an arc in $G(S)$ if such an arc exists. A formal description of the valid cycle detection algorithm is given below.

```
algorithm valid cycle detection
begin
    $\boldsymbol{P}_1 := \{(i, j) \in A(G(S)) : \alpha_{ij} < 0\}$;
    $k := 1$;
    $W^* := \emptyset; \alpha(W^*) = \infty$;
    while $k < R$ and $\alpha(W^*) \geq 0$ do
    begin
        while $\boldsymbol{P}_k \neq \emptyset$ do
        begin
            remove a path $P$ from $\boldsymbol{P}_k$
            let $i$:= $head(P)$ and $h$:= $tail(P)$
            if $(i, h) \in A(G(S))$ and $\alpha(P)+\alpha_{ih}<\alpha(W^*)$ then $W^*$:= $P\cup\{(i, h)\}$;
            for each $(i, j) \in A(i)$ do
                if $label(j) \notin label(P)$ and $\alpha(P)+\alpha_{ij} < 0$ then
                begin
                    add the path $P\cup\{(i, j)\}$ to $\boldsymbol{P}_{k+1}$
                    if $\boldsymbol{P}_{k+1}$ contains another path with the same key as
                    $P\cup\{(i, h)\}$ then remove the dominated path;
                endif;
            endfor
        endwhile;
        $k = k+1$;
    endwhile;
    return $W^*$;
end;
```

Although the algorithm above is not polynomial, it worked very well in practice for some partitioning problems on which the algorithm was tested [6,7,29].

Path Exchange as a Cyclic Exchange

Let us now consider how a valid path can be obtained if $G(S)$ has one. Note that by Lemma 20.2, every valid cycle of length r gives $r - 1$ valid paths. Sometimes path exchanges resulting from these valid paths may be better than the best cyclic exchange. Even in the absence of a valid cycle, it is possible that $G(S)$ may have valid paths that lead to an improved solution. Let S be a partition containing p subsets, and suppose that S^* is a partition obtained from S by a cyclic exchange. In general, S^* will also contain p subsets, and they will have the same cardinalities as the original subsets of S. These are limitations of the cyclic exchange. One approach to overcome this drawback is to periodically explore the possibility of splitting one or more subsets within a partition or merging two or more subsets within a partition. In the case of communication spanning tree problems, some cyclic exchange moves lead to such simple split operations in a natural way. Interestingly, path exchange moves have the natural property of the possibility of decreasing the number of subsets in a partition. By allowing the possibility of an empty set, an improvement graph can be constructed where a path exchange can even increase the number of subsets while moving from one partition to another. This variation of path exchange is called *enhanced path exchange.*

Interestingly, we now observe that path exchanges can be viewed as cyclic exchanges in a modified improvement graph $\tilde{G}(S) = \left(\tilde{V}, \tilde{A}\right)$, which is a supergraph of $G(S)$. Introduce p new nodes (called *pseudonodes*) $s_1, s_2, \ldots, s_p$ and another node v called the *origin node.* Thus the node set $\tilde{V}$of $\tilde{G}(S)$ is $E \cup \left\{s_1, s_2, \ldots, s_p, v\right\}$. The graph $\tilde{G}(S)$ contains all arcs of $G(S)$ along with additional arcs from the set $\{(v, j) : j \in E\} \cup \{(j, s_i) : i \neq \theta(j)$ and $S_i \cup \{j\}$ is a feasible subset$\} \cup \{(s_i, v) : i = 1, 2, \ldots, p\}$ (see Figure 20.2). The cost of these additional arcs are defined as follows: $\alpha_{vj} = c(S_{\theta(j)}\backslash\{j\}) - c(S_{\theta(j)})$ for all $j \in \mathsf{E}; \alpha_{s_i v} = 0$ for $i = 1, 2, \ldots, p$; and $\alpha_{js_i} = c(S_i\cup\{j\}) - c(S_i)$ for all $(j, s_i) \in \{(j, s_i) : i \neq \theta(j)$ and $S_i \cup \{j\}$ is a feasible subset$\}$.

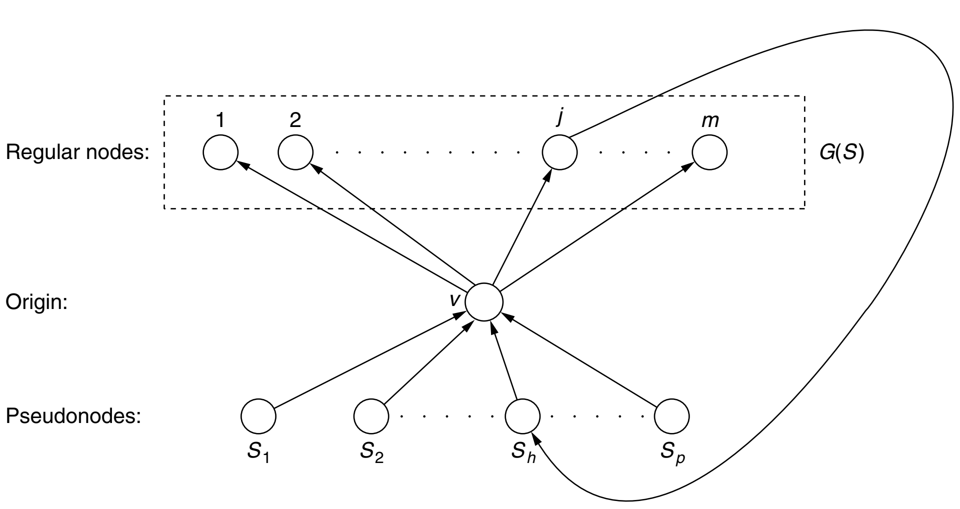


FIGURE 20.2 Construction of the improvement graph $\tilde{G}(S)$.

Note that a directed cycle $\mathbb{C}$ in $\tilde{G}(S)$ containing the node v will have exactly one pseudonode and the remaining nodes are regular. Further, these regular nodes form a path in $\mathbb{C}$ called the *regular path* associated with $\mathbb{C}$. A directed cycle $\mathbb{C}$ containing the node v in the graph $\tilde{G}(S)$ is said to be *subset disjoint* if its associated regular path is subset disjoint.

Lemma 20.3 (**Ahuja et al.** [7])

There is a one-to-one cost-preserving correspondence between path exchanges with respect to the partition S and subset-disjoint directed cycles in $\tilde{G}(S)$ containing the origin node v.

If a valid cycle in the improvement graph contains the origin node v, it corresponds to an improving path exchange. If it does not contain v, it corresponds to an improving cyclic exchange. Moreover, the improvement graph $\tilde{G}(S)$ can be further modified by adding a new node and appropriate arcs so that improving enhanced path exchanges can be identified. A VLSN search algorithm based on a multiexchange (path or cyclic) can be described as follows:

```
Algorithm Multiexchange
begin
    compute a feasible solution $S$ to the partitioning problem;
    construct the improvement graph $\tilde{G}(S)$;
    while $\tilde{G}(S)$ contains a valid cycle do
        obtain a valid cycle $W$ in $\tilde{G}(S)$;
        If $W$ contains the origin node $v$ then
            perform a path exchange using the corresponding regular path;
        else
            perform a cyclic exchange corresponding to $W$;
        endif
            update $S$ and $\tilde{G}(S)$;
    endwhile;
end.
```

Many problem-specific features can be used to simplify calculations in the VLSN search algorithm discussed above for the partitioning problem. The complexity of the construction of the improvement graph depends on that of the feasibility oracle and on the evaluation of the cost function $c(.)$. Also, problem-specific information may be used to update the improvement graph from iteration to iteration efficiently.

The Capacitated Minimum Spanning Tree Problem

Let G be a graph on the node set $V \cup \{0\}$, where $V = \{1, 2, \ldots, n\}$. Nodes in V are called *terminal nodes*, and the node 0 is called the *central node*. For each arc (i, j), a nonnegative cost c_{ij} is prescribed. Also for each node $i \in V$, a nonnegative demand $w_i$is prescribed. Let L be a given real number. For any spanning tree T of G, let $T_1, T_2, \ldots, T_{K_T}$ denote the components of T-$\{0\}$. Then the CMST problem is to

$$\begin{aligned} &\text{Minimize} \quad \sum_{ij \in T} c_{ij} \\ &\text{subject to} \\ &\qquad\qquad T \text{ is a spanning tree of } G \\ &\qquad\qquad \sum_{i \in T_j} w_i \leq L, \quad j = 1, 2, \ldots, K_T \end{aligned} \tag{20.3}$$

For any subgraph H of G, we sometimes let $V(H)$ denote its node set and let $E(H)$ denote its edge set. We also use the notation $ij \in H$ $(i \in H)$ to represent $(i, j) \in E(H)$ $(i \in V(H))$ when there is no ambiguity. It is easy to see that for an optimal spanning tree T, the component T_i must be a minimum spanning tree of the subgraph of G induced by $V(T_i)$, $i = 1, 2, \ldots, K_T$.

Thus CMST can be viewed as a partition problem where the ground E is the set V of nodes in G, the feasibility oracle is to verify the condition $\sum_{i \in S_j} w_i \leq L$. The cost $c(S_i)$ is the cost of the minimum spanning tree T_i of the subgraph of G induced by $S_i \cup \{0\}$. Without loss of generality, we assume that node 0 is a pendant node (node of degree 1) of T_i. Otherwise, we can decompose T_i into a number of subtrees equal to the degree of node 0, where 0 is a pendant node in each such subtree, yielding an alternative feasible solution.

The algorithm multiexchange can be used to find a heuristic solution for CMST. For details on implementation aspects of this heuristic specifically for the CMST problem, we refer to Ref. [6]. Instead of using the simple node exchange, Ahuja et al. [6,7] also considered subtree exchange neighborhoods and composite neighborhoods [6]. The node exchange worked well for problems where the node weights are identical (homogeneous problems). Subtree exchanges worked well for problems with different node weights (heterogeneous problems). The composite neighborhood worked well for both class of problems and produced improved solutions for several benchmark problems. Table 20.1 (taken from Ref. [6]) summarizes these results.

20.3 Extended Neighborhoods and Domination Analysis

Let us now look at some theoretical issues related to performance guarantees of VLSN search algorithms. Glover and Punnen [47] introduced the concept of domination ratio to assess the performance guarantee of a heuristic algorithm. Let α be a heuristic algorithm for a COP P. Then the domination ratio of α, denoted by $dom(\alpha)$, is

$$\mathop{Inf}_{(F,f)} |\{S \in F \ : \ f(S) \geq f(S_\alpha)\}|/|F|$$

where S_α is the solution obtained by the heuristic α on instance (F, f). Note that $0 < dom(\alpha) \leq 1$ and $dom(\alpha) = 1$ if and only if α guarantees an optimal solution.

Identifying tight deterministic bounds on domination ratio could be difficult for many algorithms. For various recent works on domination analysis of algorithms we refer to Refs.[16,30,48,49]. For VLSN search algorithms for a COP,

$$\mathop{Inf}_{(F,f)} \left\{ \frac{|N(S)|}{|F|} \ : \ S \in F \right\}$$

gives a trivial lower bound on the domination ratio, provided we can find an improving solution in $N(S)$ if exists.

TABLE 20.1 Evaluation of the Composite Neighborhood Algorithm

Problem ID	Number of Nodes	Number of Arcs	Capacity	Best Available Solution	Composite Neighborhood Solution
tc80-3	81	3,240	10	880	878
CM50-3	50	1,225	400	735	732
CM50-4	50	1,225	400	567	564
CM50-5	50	1,225	400	612	611
CM50-2	50	1,225	800	515	513
CM100-1	100	4,950	200	520	516
CM100-2	100	4,950	200	602	596
CM100-3	100	4,950	200	549	541
CM100-4	100	4,950	200	444	437
CM100-5	100	4,950	200	427	425
CM100-1	100	4,950	400	253	252
CM100-5	100	4,950	400	224	223
CM200-1	200	19,900	200	1037	1017
CM200-2	200	19,900	200	1230	1221
CM200-3	200	19,900	200	1367	1365
CM200-4	200	19,900	200	942	927
CM200-5	200	19,900	200	981	965
CM200-1	200	19,900	400	399	397
CM200-2	200	19,900	400	486	478
CM200-3	200	19,900	400	566	560
CM200-4	200	19,900	400	397	392
CM200-5	200	19,900	400	425	420
CM200-1	200	19,900	800	256	254
CM200-3	200	19,900	800	362	361
CM200-4	200	19,900	800	276	275
CM200-5	200	19,900	800	293	292
CM200-2	200	19,900	400	486	478
CM200-3	200	19,900	400	566	560
CM200-4	200	19,900	400	397	392
CM200-5	200	19,900	400	425	420
CM200-1	200	19,900	800	256	254
CM200-3	200	19,900	800	362	361
CM200-4	200	19,900	800	276	275
CM200-5	200	19,900	800	293	292

We next introduce the concept of extended neighborhoods, which can be used to find improved domination ratios for a VLSN algorithm. The value of extended neighborhoods goes beyond establishing improved domination ratios. It generalizes the concept of exact neighborhoods and provides some theoretical insights on why the VLSN search algorithms work well in practice. We will address this aspect later in this section. Let us first consider some basic definitions and properties. For details of the theory of extended neighborhoods, we refer to Ref. [50].

Consider an instance $I = (\mathsf{F}, f)$ of a COP and let N be a neighborhood function defined on it. Let L_I^N denote the collection of all locally optimal solutions of I with respect to N. Two neighborhoods N^1 and N^2 are said to be LO-equivalent for a COP P if and only if $L_I^{N^1} = L_I^{N^2}$ for all instances I of P. A neighborhood function N^* of P is called an *extended neighborhood* of N if

(i) N^* and N are LO-equivalent, and
(ii) for every neighborhood function N^0that is LO-equivalent to N, $N^0(S) \subseteq N^*(S)$ for all $S \in \mathsf{F}$ and for all $I =(\mathsf{F}, f)$ of P.

Equivalently, N^* is the largest neighborhood, that is, LO-equivalent to N for P. If $N^* = \mathsf{F}$, then N is an exact neighborhood [42]. (A neighborhood is *exact* if every local optimum is a global optimum.) In this way, the concept of an extended neighborhood generalizes the concept of exact neighborhoods.

Note that the domination ratio of any local search algorithm using the neighborhood N will be at least

$$\underset{(F,f)}{Inf}\left\{\frac{|N^*(S)|}{|F|}\ :\ S \in F\right\}$$

The well-known 2-opt neighborhood for the TSP on n nodes contains $n(n-1)/2$ elements, while the extended neighborhood of 2-opt for the TSP contains at least $(n/2)!$ elements [50].

A COP is said to have linear cost if $f(x) = cx$, where $c = (c_1, c_2, \ldots, c_m)$ is the vector element costs and x the incidence vector representing a feasible solution. Note that each feasible solution S can be represented by its incidence vector $x = (x_1, x_2, \ldots, x_m)$ where

$$x_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{if } i \notin S \end{cases}$$

We denote by F_x the collection of all incidence vectors of elements of F. When x is the incidence vector of S, we sometimes denote $N(S)$ by $N(x)$. Let $N_F(x) = \{x^1, x^2, \ldots, x^K\}$ for some K. Then for $i = 1$ to K, let $v^i = x^i - x$. We refer to v^i as a *neighborhood vector* and let $V_F^N(x)$ be an $m \times K$ matrix whose ith column is v^i. $V_F^N(x)$ is called the matrix of neighborhood vectors at x. The following theorem characterizes extended neighborhoods for COPs with a linear cost function.

Theorem 20.4 **(Orlin and Sharma [50])**

Let P *be a COP with a linear cost function and let* N *be an associated neighborhood function. Let* F *be the collection of feasible solutions of any instance and let* $V_F^N(x)$ *be the matrix of neighborhood vectors. Then the extended neighborhood* N_F^* *of* N_F *is given by* $N_F^*(x) = \left\{\hat{x} \in F_x : \hat{x} = x + V_F^N(x)\lambda;\ \lambda \geq 0 \text{ and } \lambda \in \Re^{|N_F(x)|}\right\}$.

To illustrate the characterization of extended neighborhood in Theorem 20.4, let us construct the extended neighborhood of the 2-opt neighborhood for the TSP. In the definition of COP with a linear cost function, if E is selected as the edge set of a complete graph G and if F is selected as the family of all tours in G, we get an instance of the TSP. Let the number of nodes in G be n. A tour T in G is represented as a sequence $\langle i_1, i_2, \ldots, i_n, i_1\rangle$, where $(i_k, i_{k+1}) \in T$ for $k = 1, 2, \ldots, n-1$ and $(i_n, i_1) \in T$. Given a tour $T = \langle i_1, i_2, \ldots, i_n, i_1\rangle$, a 2-opt move can be represented as a set $\{k, l\}$, where i_k and i_l are nonadjacent in T. The move $\{k, l\}$ represents the operations of removing edges (i_k, i_{k+1}), (i_l, i_{l+1}) from T and adding edges (i_k, i_l), (i_{k+1}, i_{l+1})*to* T-$\{(i_k, i_{k+1}), (i_l, i_{l+1})\}$, where the index $n + 1 = 1$. The 2-opt neighborhood of a tour T denoted by 2-opt(T) is the collection of all tours in G that can be obtained by a 2-opt move from T. Let $T^{k,l}$ be the tour obtained from T by the 2-opt move $\{k, l\}$. The neighborhood vector corresponding to this move, denoted by $v_T^{k,l}$ is given by

$$v_T^{k,l}(e) = \begin{cases} -1 & \text{if } e = (i_k, i_{k+1}) \text{ or } e = (i_l, i_{l+1}) \\ 1 & \text{if } e = (i_k, i_l) \text{ or } e = (i_{k+1}, i_{l+1}) \\ 0 & \text{otherwise} \end{cases}$$

If x is the incidence vector of T, then $x + v_T^{k,l}$ is the incidence vector of $T^{k,l}$. Thus by Theorem 20.4, we have the extended neighborhood of 2-opt(T), denoted by 2-opt*(T) is given by

$$\left\{x' : x' \text{ is a tour in } G \text{ and } x' = x + \sum_{\{k,l\} \in 2\text{-}opt} \lambda_{kl} v_T^{k,l},\ \lambda_{kl} \geq 0\right\} \tag{20.4}$$

Theorem 20.5 **(Orlin and Sharma [50])**

(i) $|2\text{-}opt^*(T)| \geq (\lceil n/2 \rceil - 3)!(n-3)$. (ii) *Finding the best tour in 2-opt*(T) *is NP-hard.*

By definition, 2-opt and 2-opt* have the same set of local optima, but the size of 2-opt*(T) is exponentially larger than the size of 2-opt. We now show that using the same starting solution, a local search with respect to 2-opt* can terminate at a solution that cannot be reached by a local search with respect to 2-opt.

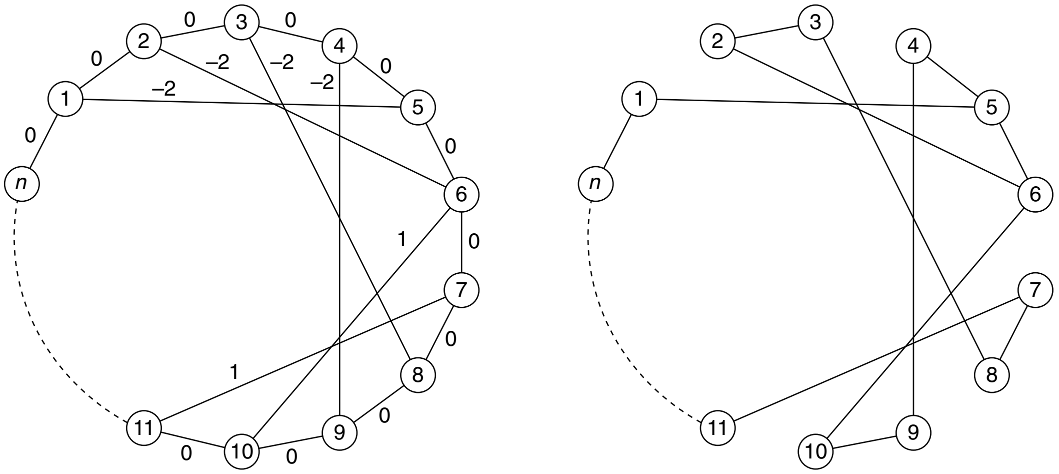


FIGURE 20.3 The graph constructed in the proof of Theorem 20.6 and tour T^*.

Theorem 20.6 (**Orlin and Sharma [50]**)

Let T be any tour with at least 11 nodes. There is a cost vector c and a tour T^* such that T^* is not reachable from T by local search with respect to 2-opt, butT^* is reachable from Tby a local search with respect to 2-opt.*

Proof

Without loss of generality assume that $T = \langle 1, 2, \ldots, n, 1\rangle$. Let all edges of T have cost 0. Introduce edges $(1, 5)$, $(2, 6)$, $(3, 8)$, and $(4, 9)$ with cost -2 and introduce edges $(6, 10)$ and $(7, 11)$ with cost 1. Complete the graph by introducing the remaining edges with a large cost (see Figure 20.3).

It can be verified that $T^* = \langle 1, 5, 4, 9, 10, 6, 2, 3, 8, 7, 11, 12, 13, \ldots, n, 1\rangle$ is an optimal tour with cost -6 . Since $T^* = T + v_T^{1,5} + v_T^{6,10} + v_T^{3,8}$, by Theorem 20.4, $T^* \in$ 2-opt*. T^*, however, is not reachable from T using improving exchanges since the moves $\{1,5\}$ and $\{3, 8\}$ lead to local optima different from T^* and $\{6, 8\}$ is not an improving move. This completes the proof.

Thus, local search with respect to extended neighborhoods fundamentally offers more possibilities than does local search using the original neighborhood. This observation further strengthens our confidence in using VLSN search algorithms to obtain good quality solutions.

20.4 Approximate Local Search

One of the most important theoretical questions in VLSN search (and local search, in general) is to find a good bound on the number of improvement steps required to reach termination. In addressing this important theoretical question, Johnson et al. [5] introduced the complexity class *PLS* [52]. A COP together with a neighborhood function belongs to class PLS if

(i) its instances are polynomial-time recognizable and an initial feasible solution is efficiently computable;
(ii) the feasibility of a proposed solution can be checked in polynomial time; and
(iii) the local optimality of a solution with respect to the neighborhood under consideration can be verified in polynomial time, and if it is not locally optimal, an improving solution can be obtained in polynomial time.

The class PLS has its own reduction scheme which leads to the class of PLS-complete problems. If a local optimum can be found in polynomial time for one of the PLS-complete problems, then a local optimum for every other PLS-complete problem can be found in polynomial time. For example, the TSP with Lin–Kernighan neighborhood [41], graph partitioning with swap neighborhood, and MAX CUT with flip neighborhood are all PLS-complete [3]. It is an outstanding open question whether there is a polynomial-time algorithm for finding a locally optimal solution for a PLS-complete problem.

In view of this open question, it is reasonable to ask whether it is possible to efficiently compute a solution "close" to a local optimum for a PLS-complete problem. This question is equally relevant for problems not in the class PLS, where the neighborhoods are searched by approximation algorithms. In this case it is interesting to see how close the solution produced is to a locally optimal solution.

A feasible solution S to an instance of a COP with neighborhood function N is said to be an *ε-local optimum* [46] if

$$\frac{f(S^*) - f(S)}{f(S)} \leq \varepsilon \text{ for all } S \in N(S)$$

where $\varepsilon > 0$. Computing an *ε-local optimum* is relatively easy [46] for problems where the cardinality of $N(S)$ is small (i.e., polynomial) for all S. Let S be a current solution and if no solution S^* exists in $N(S)$ such that $f(S^*) < f(S)/(1+\varepsilon)$ then S is an *ε-local optimum.* If such an S^* exists, then move to it and continue the search. It can be verified that the process will terminate in polynomial time (for fixed ε and integer cost function) and the resulting solution will be an *ε-local optimum.* This scheme is not applicable in many VLSN search algorithms where the neighborhood $N(S)$ is explored using heuristics.

We consider COPs with objective function f to be of the form $f(S) = \sum_{e \in S} c_e$. If there is a polynomial-time algorithm for computing an improving solution in a neighborhood, then there is a polynomial-time algorithm for computing an *ε-local optimum.* This result was originally obtained by Orlin et al. [46].

The algorithm to compute an *ε-local optimum* starts with a feasible solution S^0. Then the element costs c_e for $e \in E$ are modified using a prescribed scaling procedure to generate a modified instance. Using local search on this modified problem, we look for a solution with an objective function value (with respect to the original cost) that is half of S^0. If no such solution is found, we are at a local optimum for the modified problem and output its solution. Otherwise we replace S^0 by the solution of cost less than half and the algorithm is repeated. A formal description of the algorithm is given below. We assume that a local augmentation procedure IMPROVE$_{\mathrm{N}}$ is available, which with input of a neighborhood $N(S)$ and a cost function f computes an improved solution or declares that no improved solution exists in $N(S)$.

```
Algorithm ε-local search

Input: Objective function $f : 2^E \rightarrow \aleph$; subroutine IMPROVE$_{\mathrm{N}}$ ; initial feasible solution $S^0 \in \boldsymbol{F}$;
       accuracy $\varepsilon > 0$.
Output: Solution $S^\varepsilon \in \boldsymbol{F}$ that is an $\varepsilon$-local optimum with respect to $N$ and $f$.
Step 1: $i := 0$
Step 2: $K := f(S^i)$, $q := \frac{K\varepsilon}{2m(1+\varepsilon)}$, and $c'_e := \left\lceil \frac{c_e}{q} \right\rceil q$ for $e \in E$;
Step 3: $k := 0$ and $S^{i,k} := S^i$
Step 4: repeat
            Call IMPROVE$_{\mathrm{N}}$($S^{i,k}$, $f'$);
            {comment: $f'(S) = \sum_{e \in S} c'_e$}
            if the answer is "NO", then
                Let $S^{i,k+1} \in$ N($S^{i,k}$) such that $c'$( $S^{i,k+1}$)< $c'$( $S^{i,k}$); set $k := k + 1$;
            else $S^\varepsilon := S^{i,k}$; stop
        until $c(S^{i,k}) \leq K/2$;
Step 5: $S^{i+1}$:= $S^{i,k}$, set $i := i + 1$ and goto step 2.
```

Theorem 20.7 **(Orlin et al. [46])**

The ε-local search algorithm produces an ε-local optimum.

Proof

Let S^ε be the solution produced by the algorithm, and let S be an arbitrary solution in the neighborhood $N(S^\varepsilon)$. Let K and q denote the corresponding values from the last execution of step 2 of the algorithm. Note that

$$f(S^\varepsilon) = \sum_{e\in S^\varepsilon} c_e \le \sum_{e\in S^\varepsilon} \left\lceil \frac{c_e}{q} \right\rceil q \le \sum_{e\in S} \left\lceil \frac{c_e}{q} \right\rceil q \le \sum_{e\in S} q \left(\left\lceil \frac{c_e}{q} \right\rceil + 1 \right) \le \sum_{e\in S} c_e + mq = f(S) + mq$$

where $m = |E|$. Here, the second inequality follows from the fact that S^ε is locally optimal with respect to f'. Together with $f(S^\varepsilon) \ge K/2$, we have

$$\frac{f(S^\varepsilon) - f(S)}{f(S)} \le \frac{mq}{f(S)} \le \frac{mq}{f(S^\varepsilon) - mq} \le \frac{2mq}{K - 2mq} = \varepsilon$$

This completes the proof.

We now analyze the complexity of ε-local search algorithm as given in Ref. [46]. In each improving move within the local search in step 4 of the algorithm, the objective function value (with respect to f') is decreased by at least q units. Thus, the number of calls to IMPROVE$_{\mathrm{N}}$ between two consecutive iterations of step 2 is $O(m(1+\varepsilon)/\varepsilon) = O(m/\varepsilon)$. Step 2 is executed at most $\log f(S^0)$ times, where S^0 is the starting solution. Thus the total number of times neighborhoods searched is $O(m\varepsilon^{-1} \log f(S^0))$. Thus, whenever IMPROVE$_{\mathrm{N}}$ is a polynomial algorithm, ε-local search computes an ε-local optimum in polynomial time for fixed ε.

A strongly polynomial bound on the number of iterations was also proved in Ref. [46]. The proof makes use of the following lemma.

Lemma 20.8 (Radzik [51])

Let $d = (d_1, \ldots, d_m)$ be a real vector and let $y_1, \ldots, y_p$ be vectors on $\{0, 1\}^m$. If for all $i = 1, \ldots, p-1$, $0 \le d\, y_{i+1} \le \frac{1}{2} d y_i$ then $p = O(m \log m)$.

After each execution of step 2, K is reduced at least by half. Further, K is a linear combination of c_e for $e \in E$ with coefficients 0 or 1. Lemma 20.8 implies that step 2 of the ε-local search algorithm can be executed at most $O(m \log m)$ times. Thus IMPROVE$_{\mathrm{N}}$ at step 4 is called at most $\mathrm{O}(\varepsilon^{-1} m^2 \log m)$ times. Let $\zeta(m, \log c_{\max})$ be the complexity of IMPROVE$_{\mathrm{N}}$, $\xi(n)$ be the time needed to obtain a feasible starting solution, and $K^0 = f(S^0)$, where $c_{\max} = \max\{c_{\mathrm{e}} : e \in E\}$. Thus we have the following complexity result.

Theorem 20.9 (Orlin et al. [46])

The ε-local search algorithm correctly identifies an ε-locally optimal solution of an instance of a COP in $O(\xi(n) + \zeta(n, \log c_{max}) n\varepsilon^{-1} min\{n \log n, \log K^0\})$ time.

If the neighborhood N is exact, then ε-local search produces an ε-optimal solution [46]. Suppose that the neighborhood N is searched approximately. That is, IMPROVE$_{\mathrm{N}}$ detects an improved solution or declares that the current solution is δ-locally optimal. Even in this case an ε-local optimum can be identified in (strongly) polynomial time for any fixed ε.

20.5 Concluding Remarks

In this chapter we have discussed techniques for developing VLSN search algorithms. Empirical and theoretical indicators are provided to substantiate our belief that VLSN search algorithms are powerful tools for obtaining high-quality solutions for hard problems in reasonable amount of computational time.

Acknowledgments

We are thankful to Temel Oncan for his assistance in preparing the manuscript. This work was partially supported by NSF grant DMI-0217359 awarded to Ravi Ahuja, DMI-0238815 grant awarded to Özlem

Ergun, NSF grant DMI-0217123 awarded to James B. Orlin, and an NSERC discovery grant awarded to Abraham P. Punnen.

References

[1] Gutin, G. and Punnen, A. P., *The Traveling Salesman Problem and Its Variations*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[2] Johnson, D. S. and McGeoch, L. A., Experimental analysis of heuristics for the STSP, in *The Traveling Salesman Problem and Its Variations*, Gutin, G. and Punnen, A. P., Eds., Kluwer Academic Publishers, Dordrecht, 2002.

[3] Schaffer, A. A. and Yannakakis, M., Simple local search problems that are hard to solve, *SIAM J. Comput.*, 20, 56, 1991.

[4] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C., Optimization by simulated annealing: an experimental evaluation; Part 1, Graph partitioning. *Oper. Res.*, 37, 865, 1989.

[5] Johnson, D. S., Papadimitriou, C. H., and Yannakakis, M., How easy is local search? *J. Comput. Syst. Sci.*, 37, 79, 1988.

[6] Ahuja, R. K., Orlin, J. B., and Sharma, D., A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem, *Oper. Res. Lett.*, 31, 185, 2003.

[7] Ahuja, R. K., Orlin, J. B., and Sharma, D., Multi-exchange neighborhood search structures for the capacitated minimum spanning tree problem, *Math. Program.*, 91, 71, 2001.

[8] Sharma, D., Cyclic Exchange and Related Neighborhood Structures for Combinatorial Optimization Problems, Ph.D. thesis, Operations Research Center, MIT, Cambridge, MA, 2002.

[9] Agarwal, R., Ahuja, R. K., Laporte, G., and Shen, Z. J., A composite very large-scale neighborhood search algorithm for the vehicle routing problem, in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Leung, J. Y.-T., Ed., Chapman & Hall/CRC, London/Boca Raton, FL, 2003, chap. 49.

[10] De Franceschi, R., Fischetti, M., and Toth, P., A new ILP-based refinement heuristic for vehicle routing problems, *Math. Program.*, 105, 471–499, 2006.

[11] Ergun, Ö., New Neighborhood Search Algorithms Based on Exponentially Large Neighborhoods. Ph.D. thesis, Operations Research Center, MIT, Cambridge, MA, 2001.

[12] Ergun, Ö., Orlin, J. B., and Steele-Feldman, A., Creating Very Large-Scale Neighborhoods Out of Smaller Ones by Compounding Moves, Technical report, 2002.

[13] Deĭneko, V. G. and Woeginger, G. J., A study of exponential neighborhoods for the traveling salesman problem and the quadratic assignment problem, *Math. Program.*, 87, 519, 2000.

[14] Gutin, G., On the efficiency of a local algorithm for solving the traveling salesman problem, *Autom. Remote Control*, 49, 1514, 1988.

[15] Gutin, G., Yeo, A., and Zverovitch, A., Exponential neighborhoods and domination analysis for the TSP, in *The Traveling Salesman Problem and Its Variations*, Gutin, G. and Punnen, A. P., Eds., Kluwer Academic, Dordrecht, 2002, p. 223.

[16] Punnen, A. P., The traveling salesman problem: new polynomial approximation algorithms and domination analysis, *J. Inf. Optimization*, 22, 191, 2001.

[17] Sarvanov, V. I. and Doroshko, N. N., The approximate solution of the traveling salesman problem by a local algorithm that searches neighborhoods of exponential cardinality in quadratic time, in *Software: Algorithms and Programs*, Vol. 31, Math. Institute of the Belorussian Acad. Sci. Minsk, Minsk, 1981, p. 8 (in Russian).

[18] Ahuja, R. K., Kumar, A., Jha, K. C., and Orlin, J. B., Exact and Heuristic Algorithms for the Weapon-Target Assignment Problem, Technical report, 2003.

[19] Yagiura, M., Ibaraki, T., Glover, F., An ejection chain approach for the generalized assignment problem, *INFORMS J. Comput.*, 16, 133, 2004.

[20] Yagiura, M., Iwasaki, S., Ibaraki, T., and Glover, F., A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem, *Disc. Optimization*, 1, 87, 2004.

[21] Yagiura, M., Ibaraki, T., and Glover, F., A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, 169, 548–569, 2006.

[22] Ahuja, R. K., Orlin, J. B., Pallottino, S., Scaparra, M. P., and Scutella, M. G., A multi-exchange heuristic for the single source capacitated facility location, *Manage. Sci.*, 50, 749, 2004.

[23] Agarwal, R., Ergun, Ö., Orlin, J. B., and Potts, C. N., Solving Parallel Machine Scheduling Problems with Variable Depth Local Search, Technical report, 2004.

[24] Ahuja, R. K., Goodstein, J., Liu, J., Mukherjee, A., and Orlin, J. B., Solving the combined through-fleet assignment model with time windows using neighborhood search, *Networks*, 44, 160, 2004.

[25] Ahuja, R. K., Goodstein, J., Liu, J., Mukherjee, A., Orlin, J. B., and Sharma, D., Solving multi-criteria combined through-fleet assignment model, in *Operations Research in Space and Air*, Ciriani, T. A., Fasano, G., Gliozzi, S., and Tadei, R., Eds., Kluwer Academic Publishers, Dordrecht, 2003, p. 233.

[26] Ahuja, R. K., Goodstein, J., Mukherjee, A., Orlin, J. B., and Sharma, D., A Very Large-Scale Neighborhood Search Algorithm for the Combined Through-Fleet Assignment Model, Technical report, 2001.

[27] Ahuja, R. K., Jha, K. C., Orlin, J. B., and Sharma, D., Very Large Scale Neighborhood Search Algorithm for the Quadratic Assignment Problem, Technical report, 2002.

[28] Ropke, S. and Pisinger, D., An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows, Technical report, 2004.

[29] Cunha, C. B. and Ahuja, R. K., Very Large-Scale Neighborhood Search for the K-Constrained Multiple Knapsack Problem, Technical report, 2004.

[30] Huang, W., Romeijn, H. E., and Geunes, J., The continuous-time single-sourcing problem with capacity expansion opportunities, *Nav. Res. Logistics*, 52, 193, 2005.

[31] Ahuja, R. K., Very Large-Scale Neighborhood Search Algorithms for Minimizing the Beam-On Time and Set-Up Time in Radiation Therapy Treatment Planning, Working paper, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL, 2002.

[32] Avella, P., D'Auria, B., Salerno, S., and Vasil'ev, I., Computational Experience with Very Large-Scale Neighborhood Search for High-School Timetabling, Technical report, 2004.

[33] Chiarandini, M., Dumitrescu, I., and Stützle, T., Local Search for the Colouring Graph Problem. A Computational Study, TR AIDA-03-01, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 2003.

[34] Glover, F. and Laguna, M., *Tabu Search*, Kluwer Academic Press, Dordrecht, The Netherlands, 1997.

[35] Michalewicz, Z., *Genetic Algorithms + Data Structure = Evolution Programs*, 3rd ed., Springer, Berlin, 1996.

[36] Rego, C. and Alidaee, B., *Metaheuristic Optimization via Memory and Evolution*, Kluwer Academic Publishers, Dordrecht, 2005.

[37] Feo, T. A. and Resende, M. G. C., Greedy randomized adaptive search procedures, *J. Global Optimization*, 6, 109, 1995.

[38] Thompson, P. M. and Psaraftis, H. N., Cyclic transfer algorithms for multi-vehicle routing and scheduling problems, *Oper. Res.*, 41, 935, 1993.

[39] Vincent, T. and Kwan, M. C., Adapting the large neighborhood search to effectively solve pickup and delivery problems with time windows, *International Conference on Tools with Artificial Intelligence (ICTAI)*, 2004, p. 519.

[40] Glover, F., Ejection chains, reference structures and alternating path methods for traveling salesman problems, *Discrete Appl. Math.*, 65, 223, 1996.

[41] Lin, S. and Kernighan, B. W., An effective heuristic algorithm for the traveling salesman problem, *Oper. Res.*, 21, 972, 1973.

[42] Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, New York, 1982.

[43] Thompson, P. M. and Orlin, J. B., The Theory of Cyclic Transfers, Working paper OR200-89, Operations Research Center, MIT, Cambridge, MA, 1989.

[44] Gavish, B., Topological design telecommunications networks—local access design methods, *Ann. Oper. Res.*, 33, 17, 1991.

[45] Amberg, A., Domschke, W., and Voß, S., Capacitated minimum spanning trees: algorithms using intelligent search, *Comb. Optimization: Theor. Pract.*, 1, 9, 1996.

[46] Orlin, J. B., Punnen, A. P., and Schulz, A. S., Approximate local search in combinatorial optimization, *SIAM J. Comput.*, 33, 1201, 2004.

[47] Glover, F. and Punnen, A. P., The traveling salesman problem: new solvable cases and linkages with the development of approximation algorithms, *J. Oper. Res. Soc.*, 48, 502, 1997.

[48] Punnen, A. P. and Kabadi, S. N., Domination analysis of some heuristics for the asymmetric traveling salesman problem, *Discrete Appl. Math.*, 119, 117, 2002.

[49] Punnen, A. P., Margot, F., and Kabadi, S. N., TSP heuristics: domination analysis and complexity, *Algorithmica*, 35, 111, 2003.

[50] Orlin, J. B. and Sharma, D., Extended neighborhood: definition and characterization, *Math. Prog., Ser A.*, 101, 537, 2004.

[51] Radzik, T., Parametric flows, weighted means of cuts, and fractional combinatorial optimization, in *Complexity in Numerical Optimization*, Pardalos, P., Ed., World Scientific, Hackensack, NJ, 1993, p. 351.

[52] Aarts, E. M. L. and Lenstra, J. K., *Local Search in Combinatorial Optimization*, Wiley, New York, 1979.

[53] Ahuja, R. K., Ergun, Ö., Orlin, J. B., Punnen, A. P., A survey of very large-scale neighborhood search techniques, *Discrete Appl. Math.*, 23, 75, 2001.

[54] Ahuja, R. K., Boland, N., and Dumitriscue, I., Exact and Heuristic Algorithms for the Subset-Disjoint Minimum Cost Cycle Problems, Working paper, Industrial and Systems Engineering, University of Florida, Gainesville, FL, 2001.

[55] Ahuja, R. K., Liu, J., Orlin, J. B., Goodstein, J., and Mukherjee, A., Solving the combined through-fleet assignment model with time windows using neighborhood search, *Networks*, 44, 160, 2004.

[56] Ergun, Ö. and Orlin, J. B., Very Large-Scale Neighborhood Search Based on Solving Restricted Dynamic Programming Recursions, Technical report, 2004.

[57] Ergun, Ö. and Orlin, J. B., Fast neighborhood search for the single machine total weighted tardiness problem, *Oper. Res. Lett.*, 34, 41–45, 2006.

[58] Ibaraki, T., Imahori, S., Kubo, M., Masuda, T., Uno, T., and Yagiura, M., Effective local search algorithms for routing and scheduling problems with general time-window constraints, *Transp. Sci.*, 39, 206, 2005.

[59] Krentel, M. W., Structure in locally optimal solutions, *Proc. of FOCS*, 1989, p. 216.

[60] Papadimitriou, C. H., The complexity of the Lin–Kernighan heuristic for the traveling salesman problem, *SIAM J. Comput.*, 21, 450, 1992.

[61] Papadimitriou, C. H. and Steiglitz, K., On the complexity of local search for the traveling salesman problem, *SIAM J. Comput.*, 6, 76, 1977.

[62] Sharaiha, Y. M., Gendreau, M., Laporte, G., and Osman, I. H., A tabu search algorithm for the capacitated shortest spanning tree problem, *Networks*, 29, 161, 1997.

21

Reactive Search: Machine Learning for Memory-Based Heuristics

Roberto Battiti
University of Trento

Mauro Brunato
University of Trento

21.1 Introduction: The Role of the User in Heuristics

Most state-of-the-art heuristics are characterized by a certain number of choices and free parameters, whose appropriate setting is a subject that raises issues of research methodology [1–3].

In some cases, these parameters are tuned through a feedback loop that *includes the user as a crucial learning component*: depending on preliminary algorithm tests some parameter values are changed by the user, and different options are tested until acceptable results are obtained. Therefore, the quality of results is not automatically transferred to different instances and the feedback loop can require a lengthy "trial-and-error" process every time the algorithm has to be tuned for a new application.

Parameter tuning is therefore a crucial issue both in the scientific development and in the practical use of heuristics. In some cases the role of the user as an intelligent (learning) part makes the reproducibility of heuristic results difficult and, as a consequence, the competitiveness of alternative techniques depends in a crucial way on the user's capabilities.

Reactive Search advocates the use of simple subsymbolic machine learning to automate the parameter tuning process and make it an integral (and fully documented) part of the algorithm.

If learning is performed online, task-dependent and local properties of the configuration space can be used by the algorithm to determine the appropriate balance between *diversification* (looking for better solutions in other zones of the configuration space) and *intensification* (exploring more intensively a small but promising part of the configuration space). In this way a single algorithm maintains the flexibility to deal with related problems through an internal feedback loop that considers the previous history of the search.

In the following, we shall call *reaction* the act of modifying some algorithm parameters in response to the search algorithm's behavior *during* its execution, rather than between runs. Therefore, a *reactive* *heuristic* is a technique with the ability of tuning some important parameters during execution by means of a machine learning mechanism.

It is important to notice that such heuristics are intrinsically history-dependent; thus, the practical success of this approach in some cases raises the need of a sounder theoretical foundation of non-Markovian search techniques.

21.1.1 Machine Learning for Automation and Full Documentation

Parameter tuning is a typical "learning" process where experiments are designed in a focused way, with the support of statistical estimation (parameter identification) tools.

Because of its familiarity with algorithms, the Computer Science (CS) community masters a very powerful tool for describing processes so that they can be reproduced even by a (mechanical) computer. In particular, the Machine Learning community, with significant influx from Statistics, developed in the last decades has a rich variety of "design principles" that can be used to develop machine learning methods and algorithms.

It is therefore appropriate to consider whether some of these design principles can be profitably used in the area of parameter tuning for heuristics. The long-term goal is that of completely *eliminating the human intervention* in the tuning process. This does not imply higher unemployment rates in the CS community, on the contrary, the researcher is now loaded with a heavier task: the algorithm developer must aim at transferring his expertise into the algorithm itself, a task that requires the *exhaustive description of the tuning phase* in the algorithm.

Let us note that the algorithm "complexity" will increase as a result of the process, but the price is worth paying if the two following objectives are reached:

Complete and unambiguous documentation. The algorithm (and the research paper based on the algorithm) becomes self-contained and its quality can be judged independently from the designer or specific user. This requirement is particularly important from the scientific point of view, where objective evaluations are crucial. The recent introduction of software archives (in some cases related to scientific journals) further simplifies the test and *simple reuse* of heuristic algorithms.

Automation. The time-consuming tuning phase is now substituted by an automated process. Let us note that only the final user will typically benefit from an automated tuning process. On the contrary, the algorithm designer faces a longer and harder development phase, with a possible preliminary phase of exploratory tests, followed by the above-described exhaustive documentation of the tuning process when the algorithm is presented to the scientific community.

Although formal learning frameworks do exist in the CS community, notably the Probably Approximately Correct (PAC) learning model [4,5], one should not reach the conclusion that these models can be simply adapted to the new context. On the contrary, the theoretical framework of computational learning theory and machine learning is very different from that of heuristics. For example, the definition of a "quality" function against which the learning algorithm has to be judged is complex. In addition, the abundance of negative results in computational learning should warn about excessive hopes.

Nonetheless, as a first step, some of the principles and methodologies used in machine learning can be used in an analogic fashion to develop "reactive heuristics."

21.1.2 Asymptotic Results Are Irrelevant for Optimization

Scientists, and also final users, might feel uneasy working with non-Markovian techniques because they do not benefit from the deep and wide theoretical background that covers Markovian algorithms. However, asymptotic convergence results of many Markovian search algorithms, such as Simulated Annealing (SA) [6], are often irrelevant for their application to optimization. As an example, a comparison of SA and Reactive Search has been presented in Refs. [7,8].

In any finite-time approximation, one must resort to approximations of the asymptotic convergence. In SA, for instance, the "speed of convergence" to the stationary distribution is determined by the second largest eigenvalue of the transition matrix. The number of transitions is at least *quadratic* in the size of the solution space [9], which is typically exponential in n.

When using a time-varying temperature parameter, it can happen (e.g., the Traveling Salesman Problem (TSP)) that the complete enumeration of all solutions would take less time than approximating an optimal solution with arbitrary precision by SA [9].

In addition, repeated local search [10] and even random search [11] have better asymptotic results. According to Ref. [9] "approximating the asymptotic behavior of SA arbitrarily closely requires a number of transitions that for most problems is typically larger than the size of the solution space [. . .] Thus, the SA algorithm is clearly unsuited for solving combinatorial optimization problems to optimality." Of course, practical utility of SA has been shown in many applications, in particular with fast cooling schedules, but then the asymptotic results are not directly applicable. The optimal finite-length annealing schedules obtained on specific simple problems do not always correspond to those intuitively expected from the limiting theorems [12].

21.2 Reactive Search Applied to Tabu Search

In this section, we illustrate the potential of Reactive Search by installing a reaction mechanism on the prohibition period T of a Tabu Search (TS) [13] algorithm. For the complete description of TS, the reader is referred to Chapter 23 of this handbook.

21.2.1 Prohibition-Based Diversification: Tabu Search

The TS metaheuristic is based on the use of *prohibition-based* techniques and "intelligent" schemes as a complement to basic heuristic algorithms like local search, with the purpose of guiding the basic heuristic *beyond local optimality*. It is difficult to assign a precise date of origin to these principles. For example, ideas similar to those proposed in TS can be found in the *denial* strategy of Ref. [14] (once common features are detected in many suboptimal solutions, they are forbidden) or in the opposite *reduction* strategy of Ref. [15] (in an application to the TSP, all edges that are common to a set of local optima are fixed). In very different contexts, prohibition-like strategies can be found in *cutting planes* algorithms for solving integer problems through their Linear Programming relaxation (inequalities that cut off previously obtained fractional solutions are generated) and in branch and bound algorithms (subtrees are not considered if the leaves cannot correspond to better solutions). For many examples of such techniques, see Ref. [16].

The renaissance and full blossoming of "intelligent prohibition-based heuristics" starting from the late 1980s is greatly due to the role of F. Glover in the proposal and diffusion of a rich variety of metaheuristic tools [13,17], but see also Ref. [18] for an independent seminal paper. A growing number of TS-based algorithms has been developed in the last years and applied with success to a wide selection of problems [19]. It is therefore difficult, if not impossible, to characterize a "canonical form" of TS, and classifications tend to be short-lived. Nonetheless, at least two aspects characterize many versions of TS: the fact that TS is used to complement local (neighborhood) search, and the fact that the main modifications to local search are obtained through the *prohibition* of selected moves available at the current point. TS acts to continue the search beyond the first local minimizer without wasting the work already executed, as it is the case if a new run of local search is started from a new random initial point, and to enforce appropriate amounts of diversification to avoid that the search trajectory remains confined near a given local minimizer.

In our opinion, the main competitive advantage of TS with respect to alternative heuristics based on local search like SA [6] lies in the intelligent use of the past history of the search to influence its future steps.

For a generic search space $\mathcal{X}$, let $X^{(t)} \in \mathcal{X}$ be the current configuration and $N(X^{(t)}) \subseteq \mathcal{X}$ its neighborhood. In prohibition-based search (TS) some of the neighbors can be *prohibited*, and let the subset $N_A(X^{(t)}) \subseteq N(X^{(t)})$ contain the *allowed* ones. The general way of generating the search trajectory that we consider is given by

$$X^{(t+1)} = \text{Best-Neighbor}\left(N_A\left(X^{(t)}\right)\right) \tag{21.1}$$

$$N_A(X^{(t+1)}) = \text{Allow}\left(N\left(X^{(t+1)}\right), X^{(0)}, \ldots, X^{(t+1)}\right) \tag{21.2}$$

The set-valued function ALLOW selects a subset of $N\big(X^{(t+1)}\big)$ in a manner that depends on the search trajectory $X^{(0)}, \ldots, X^{(t+1)}$.

This general framework allows several specializations. In many cases, the dependence of ALLOW on the entire search trajectory introduces too many constraints on the next move, causing the search path to avoid an otherwise promising area, or even prohibiting all neighbors. It is therefore advisable to reduce the amount of constraints by limiting the ALLOW function to the latest T configurations (where the parameter T is often called the *prohibition period*), so that Eq. (21.2) becomes

$$N_A\big(X^{(t+1)}\big) = \text{ALLOW}\left(N\big(X^{(t+1)}\big), X^{(t')}, \ldots, X^{(t+1)}\right), \quad t' = \max\{0, t - T + 1\} \qquad (21.3)$$

A practical example for Eq. (21.3) is a function ALLOW that forbids all moves that have been performed within the last T iterations. For instance, let us assume that the feasible search space $\mathcal{X}$ is the set of binary strings with a given length $L : \mathcal{X} = \{0, 1\}^L$ (this case shall be considered also in the example of Section 21.4). In this case, a practical neighborhood of configuration $X^{(t)}$ is given by the L configurations that differ from $X^{(t)}$ by a single entry. In such a case, a simple prohibition scheme may allow a move if and only if it changes an entry which has remained fixed for the previous T iterations. In other words, after an entry has been changed, it shall remain *frozen* for the following T steps.

It is apparent that the choice of the right prohibition period T is crucial to balance the amount of *intensification* (small T) and *diversification* (large T).

21.2.2 Reaction on Tabu Search Parameters

Some problems arising in TS that have been investigated in Reactive Search papers are

1. the determination of an appropriate prohibition T for the different tasks,
2. the robustness of the technique for a wide range of different problems, and
3. the adoption of minimal computational complexity algorithms for using the search history.

The three issues are briefly discussed in the following sections, together with the reaction-based methods proposed to deal with them.

21.2.2.1 Self-Adjusted Prohibition Period

In Reactive Tabu Search (RTS), i.e., Reactive Search applied to TS, the *prohibition period* T is determined through feedback (i.e., *reactive*) mechanisms during the search. At the beginning, we let $T = 1$ (the inverse of a given move is prohibited only at the next step). During the search, T increases only when there is *evidence* that diversification is needed, and it decreases when this evidence disappears. In detail: the evidence that diversification is needed is signaled by the repetition of previously visited configurations. For this purpose, all configurations found during the search are stored in memory. After a move is executed, the algorithm checks whether the current configuration has already been found and reacts accordingly (T increases if a configuration is repeated, T decreases if no repetitions occurred during a sufficiently long period).

By means of this self-adjustment algorithm, T is not fixed during the search, but it is determined in a dynamic way depending on the *local structure* of the search space. This is particularly relevant for "inhomogeneous" tasks, where the statistical properties of the search space vary widely in the different regions (in these cases a fixed T would be inappropriate).

An example of the behavior of T during the search is illustrated in Figure 21.1, for a Quadratic Assignment Problem task [20]. T increases in an exponential way when repetitions are encountered, it decreases in a gradual manner when repetitions disappear.

21.2.2.2 The Escape Mechanism

The basic tabu mechanism based on prohibitions is not sufficient to avoid long cycles. As an example, when operating on binary strings of length L, the prohibition T must be less than the length of the string, otherwise all moves are eventually prohibited; therefore, cycles longer than 2 × L are still possible. In addition, even if "limit cycles" (endless cyclic repetitions of a given set of configurations) are avoided, the

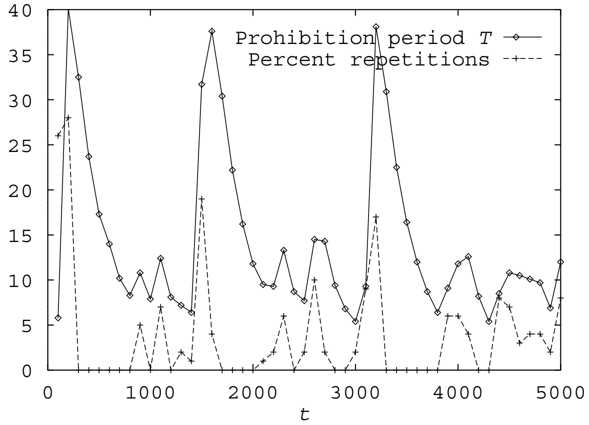


FIGURE 21.1 Dynamics of the prohibition period T on a QAP task.

first reactive mechanism is not sufficient to guarantee that the search trajectory is not confined in a limited region of the search space. A "chaotic trapping" of the trajectory in a limited portion of the search space is still possible (the analogy is with *chaotic attractors* of dynamical systems, where the trajectory is confined in a limited portion of the space, although a limit cycle is not present).

For both reasons, to increase the robustness of the algorithm a second more radical diversification step (*escape*) is needed. The *escape* phase is triggered when too many configurations are repeated too often [20]. A simple *escape* consists of a number of random steps executed starting from the current configuration (possibly with a bias toward steps that bring the trajectory away from the current search region).

With a stochastic escape, one can easily obtain the *asymptotic convergence* of RTS: in fact, in a finite search space *escape* is activated infinitely often; if the probability for a point to be reached after escaping is different from zero for all points, eventually all points will be visited—clearly including the globally optimal points. The detailed investigation of the asymptotic properties and finite-time effects of different *escape* routines to enforce long-term diversification is an open research area.

21.2.3 Implementation of History-Sensitive Techniques

The efficiency and competitiveness of history-based reaction mechanisms strongly depend on the detailed data structures used in the algorithms and on the consequent realization of the needed operations. Different data structures can possess widely different computational complexities so that attention should be spent on this subject before choosing a version of Reactive Search that is efficient on a particular problem.

Reactive-TS can be implemented through a simple list of visited configurations, or with more efficient hashing [21,20] or radix tree [20] techniques. At a finer level of detail, hashing can be realized in different ways. If the entire configuration is stored (see also Figure 21.2) an exact answer is obtained from the memory lookup operation (a repetition is reported if and only if the configuration has been visited before). On the contrary, if a "compressed" item is stored, like a hashed value of a limited length derived from the configuration, the answer will have a limited probability of *false positives* (a repetition can be reported even if the configuration is new, because the compressed items are equal by chance—an event called "collision"). Experimentally, small collision probabilities do not have statistically significant effects on the use of Reactive-TS as a heuristic tool, and hashing versions that need only a few bytes per iteration can be used.

21.2.3.1 Fast Algorithms for Using the Search History

The storage and access of the past events is executed through the well-known hashing or radix-tree techniques in a CPU time that is approximately *constant* with respect to the number of iterations. Therefore, the overhead caused by the use of the history is negligible for tasks requiring a nontrivial number of operations to evaluate the cost function in the neighborhood.

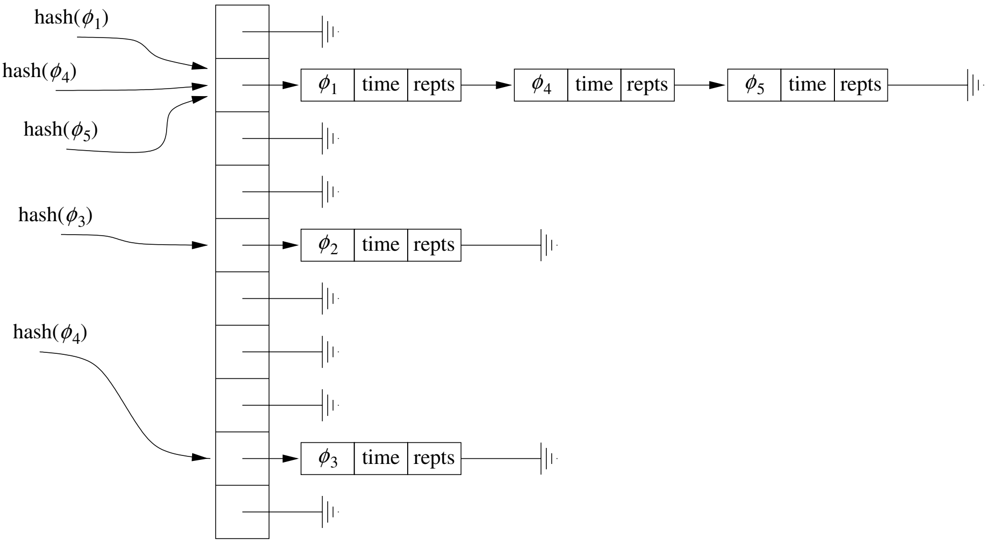


FIGURE 21.2 Open hashing scheme: items (configuration or compressed hashed value) are stored in "buckets." The index of the bucket array is calculated from the configuration.

An example of a memory configuration for the hashing scheme is shown in Figure 21.2. From the current configuration ϕ one obtains an index into a "bucket array." The items (configuration or hashed value or derived quantity, last time of visit, and total number of repetitions) are then stored in linked lists starting from the indexed array entry. Both storage and retrieval require an approximately constant amount of time if (i) the number of stored items is not much larger than the size of the bucket array and (ii) the *hashing function* scatters the items with a uniform probability over the different array indices. More precisely, given a hash table with m slots that stores n elements, a load factor $\alpha = n/m$ is defined. If collisions are resolved by chaining searches take $O(1 + \alpha)$ time, on average.

21.2.3.2 Persistent Dynamic Sets

Persistent dynamic sets are proposed to support memory-usage operations in history-sensitive heuristics in Refs. [22,23].

Ordinary data structures are *ephemeral* [24] because when a change is executed the previous version is destroyed. Now, in many contexts like computational geometry, editing, implementation of very high-level programming languages, and, last but not least, the context of history-based heuristics, multiple versions of a data structure must be maintained and accessed. In particular, in heuristics one is interested in *partially persistent* structures, where all versions can be accessed but only the newest version (the *live* nodes) can be modified. A review of ad hoc techniques for obtaining persistent data structures is given in Ref. [24] that is dedicated to a systematic study of persistence, continuing the previous work of Ref. [25].

Hashing Combined with Persistent Red-Black Trees

The basic observation is that, because TS is based on local search, configuration $X^{(t+1)}$ differs from configuration $X^{(t)}$ only because of the addition or subtraction of a single index (a single bit is changed in the string). Let us define the operations INSERT(i) and DELETE(i) for inserting and deleting a given index i from the set. As cited above, configuration X can be considered as a set of indices in $[1, L]$ with a possible realization as a balanced red-black tree (see Refs. [26,27] for two seminal papers about red-black trees). The binary string can be immediately obtained from the tree by visiting it in symmetric order, in time $O(L)$. INSERT(i) and DELETE(i) require $O(\log L)$ time, while at most a single node of the tree is allocated or deallocated at each iteration. Rebalancing the tree after insertion or deletion can be done in $O(1)$ rotations and $O(\log L)$ color changes [28]. In addition, the amortized number of color changes per update is $O(1)$ (see, e.g., Ref. [29]).

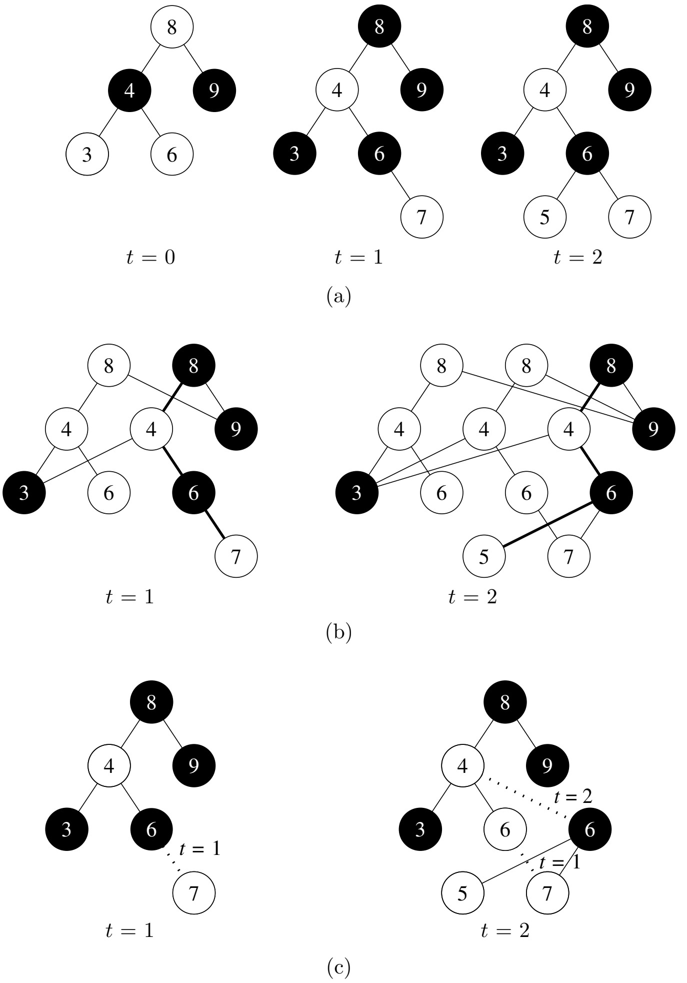


FIGURE 21.3 How to obtain (a) a partially persistent red-black tree from an ephemeral one, containing indices 3, 4, 6, 8, 9 at $t = 0$, with subsequent insertion of 7 and 5; a persistent red-black tree with path copying, with thick lines marking the copied part; (c) a persistent red-black tree with limited node copying, with broken lines denoting the "extra" pointers with time stamp.

Now, the Reverse Elimination Method [13,17,30] (a technique for the storage and analysis of the ordered list of all moves performed throughout the search) is closely reminiscent of a method studied in Ref. [25] to obtain partial persistence, in which the entire update sequence is stored and the desired version is rebuilt from scratch each time an access is performed, while a systematic study of techniques with better space–time complexities is present in Refs. [24,31]. Let us now summarize from Ref. [31] how a partially persistent red-black tree can be realized. An example of the realizations that we consider is presented in Figure 21.3.

The trivial way is that of keeping in memory all copies of the ephemeral tree (see Figure 21.3(a)), each copy requiring $O(L)$ space. A smarter realization is based on *path copying*, independently proposed by many researchers (see Ref. [31] for references). Only the path from the root to the nodes where changes are made is copied: a set of search trees is created, one per update, having different roots but *sharing* common subtrees. The time and space complexities for INSERT(i) and DELETE(i) are now of $O(\log L)$.

The method that we will use is a space-efficient scheme requiring only linear space proposed in Ref. [31]. The approach avoids copying the entire access path each time an update occurs. To this end, each node contains an additional "extra" pointer (beyond the usual left and right ones) with a time stamp.

When attempting to add a pointer to a node, if the extra pointer is available, it is used and the time of the usage is registered. If the extra pointer is already used, the node is copied, setting the initial left and right pointers of the copy to their latest values. In addition, a pointer to the copy is stored in the last parent of the copied node. If the parent has already used the extra pointer, the parent, too, is copied. Thus copying proliferates through successive ancestors until the root is copied or a node with a free extra pointer is encountered. Searching the data structure at a given time t in the past is easy: after starting from the appropriate root, if the extra pointer is used the pointer to follow from a node is determined by examining the time stamp of the extra pointer and following it iff the time stamp is not larger than t. Otherwise, if the extra pointer is not used, the normal left–right pointers are considered. Note that the pointer direction (left or right) does not have to be stored: given the search tree property it can be derived by comparing the indices of the children with that of the node. In addition, colors are needed only for the most recent (live) version of the tree. In Figure 21.3 `null` pointers are not shown, colors are correct only for the live tree (the nodes reachable from the rightmost root), extra pointers are dashed and time-stamped.

The worst-case time complexity of INSERT(i) and DELETE(i) remains of $O(\log L)$, but the important result derived in Ref. [31] is that the amortized space cost per update operation is $O(1)$. Let us recall that the total amortized space cost of a sequence of updates is an upper bound on the actual number of nodes created.

Let us now consider the context of history-based heuristics. Contrary to the popular usage of persistent dynamic sets to search past versions at a specified time t, one is interested in checking whether a configuration has already been encountered in the previous history of the search, at *any* iteration.

A convenient way of realizing a data structure supporting X-SEARCH(X) is to combine *hashing* and *partially persistent dynamic sets* (see Figure 21.4). From a given configuration X an index into a "bucket array" is obtained through a hashing function, with a possible incremental evaluation in time $O(1)$. Collisions are resolved through chaining: starting from each bucket header there is a linked list containing a pointer to the appropriate root of the persistent red-black tree and satellite data needed by the search (time of configuration, number of repetitions).

As soon as configuration $X^{(t)}$ is generated by the search dynamics, the corresponding persistent red-black tree is updated through INSERT(i) or DELETE(i). Let us now describe X-SEARCH($X^{(t)}$): the hashing value is computed from $X^{(t)}$ and the appropriate bucket searched. For each item in the linked list the

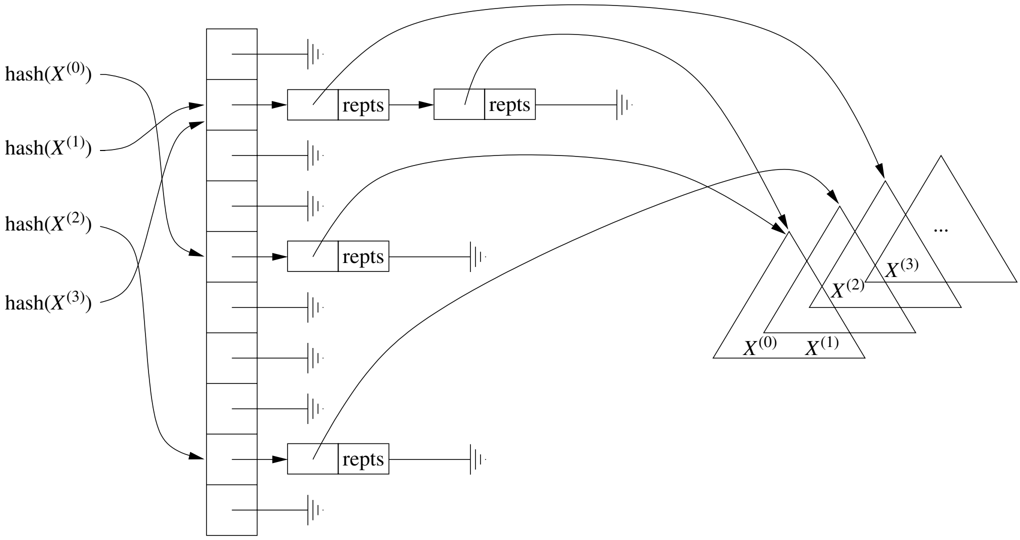


FIGURE 21.4 Open hashing scheme with persistent sets: a pointer to the appropriate root for configuration $X^{(t)}$ in the persistent search tree is stored in a linked list at a "bucket." Items on the list contain satellite data. The index of the bucket array is calculated from the configuration through a hashing function.

pointer to the root of the past version of the tree is followed and the old set is compared with $X^{(t)}$. If the sets are equal, a pointer to the item on the linked list is returned. Otherwise, after the entire list has been scanned with no success, a `null` pointer is returned.

In the last case a new item is linked in the appropriate bucket with a pointer to the root of the live version of the tree (X-INSERT(X, t)). Otherwise, the last visit time t is updated and the repetition counter is incremented.

After collecting the above cited complexity results, and assuming that the bucket array size is equal to the maximum number of iterations executed in the entire search, it is straightforward to conclude that each iteration of *reactive-TS* requires $O(L)$ average-case time and $O(1)$ amortized space for storing and retrieving the past configurations and for establishing prohibitions.

In fact, both the hash table and the persistent red-black tree require $O(1)$ space (amortized for the tree). The worst-case time complexity per iteration required to update the current $X^{(t)}$ is $O(\log L)$, the average-case time for searching and updating the hashing table is $O(1)$ (in detail, searches take time $O(1 + \alpha)$, α being the load factor, in our case upper bounded by 1). The time is therefore dominated by that required to compare the configuration $X^{(t)}$ with that obtained through X-SEARCH($X^{(t)}$), i.e., $O(L)$ in the worst case. Because $\Omega(L)$ time is needed during the neighborhood evaluation to compute the f values, the above complexity is optimal for the considered application to history-based heuristics.

21.3 Wanted: A Theory of History-Sensitive Heuristics

Randomized Markovian local search algorithms have enjoyed a long period of scientific and applicative excitement, in particular see the flourishing literature on SA [6,32]. SA generates a *Markov chain*: the successor of the current point is chosen stochastically, with a probability that does not depend on the previous history (standard SA does not learn). A consequence is that the "trapping" of the search trajectory in an attractor cannot be avoided: the system has no memory and cannot detect that the search is localized. Incidentally, the often cited asymptotic convergence results of SA are unfortunately irrelevant for the application of SA to optimization. In fact, repeated local search [10], and even random search [11] has better asymptotic results.

History-sensitive techniques in local search contain an internal *feedback loop* that uses the information derived from the past history to influence the future behavior. In the cited prohibition-based diversification techniques one can, for example, decide to increase the diversification when configurations are encountered again along the search trajectory [20,33]. It is of interest that state-of-the-art versions of SA incorporate "temporal memory" [34]. The non-Markovian property is a mixed blessing: it permits heuristic results that are much better in many cases, but makes the theoretical analysis of the algorithm difficult.

Therefore, one has an unfortunate chasm: on one side there is an abundance of mathematical results derived from the theory of Markov processes, but their relevance to optimization is dubious, on the other there is mounting evidence that simple *machine learning* or history-sensitive schemes can augment the performance of heuristics in a significant way, but the theoretical instruments to analyze history-sensitive heuristics are lacking.

The practical success of history-sensitive techniques should motivate new search streams in Mathematics and Computer Science for their theoretical foundation.

21.4 Applications of Reactive Search and the Maximum Clique Example

Reactive Search principles have been used for example for the problem of Quadratic Assignment [20], training neural nets and control problems [35], vehicle-routing problems [36–38], structural acoustic control problems [39], special-purpose VLSI realizations [40], graph partitioning [41], electric power distribution [42], maximum satisfiability [43], constraint satisfaction [44,45], optimization of continuous functions [46,47], traffic grooming in optical networks [48], maximum clique [33], real-time dispatch of trams in storage yards [49], and increasing Internet capacity [50]. Because of space limitation we consider with some detail only the application to the Maximum Clique (MC) problem.

The MC problem is NP-hard, and strong negative results have been shown about its approximability [51,52]. In particular, if P $\neq$ NP, MC is *not approximable* within $n^{1/4-\epsilon}$ for any $\epsilon > 0$, n being the number of nodes in the graph [53], and it is not approximable within $n^{1-\epsilon}$ for any $\epsilon > 0$, unless coRP = NP [54].

These theoretical results stimulated a research effort to design efficient heuristics for this problem, and computational experiments to demonstrate that optimal or close approximate values can be efficiently obtained for significant families of graphs [55,56].

In particular, a new *reactive* heuristic (Reactive Local Search (RLS)) is proposed for the MC problem in Ref. [33]. The present description is a summarized version of the cited paper.

The experimental efficacy and efficiency [33] of RLS is strengthened by an analysis of the complexity of a single iteration. It is possible to show that the worst-case cost is $O(\max\{n, m\})$, where n and m are the number of nodes and edges, respectively. In practice, the cost analysis is pessimistic and the measured number of operations tends to be a small constant times the average degree of nodes in $\overline{G}$, the complement of the original graph.

21.4.1 Reactive Local Search for Max Clique

The RLS algorithm for the MC problem takes into account the particular neighborhood structure of MC. This is reflected in the following two facts: a single reactive mechanism is used to determine the prohibition parameter T, and an explicit restart scheme is added so that all possible configurations will eventually be visited, even if the search space is not connected by using the basic local search moves. Both building blocks of RLS use the past history of the search (set of visited configurations) to influence the choice.

The admissible search space $\mathcal{X}$ is the set of all cliques in a graph G defined over a vertex set V. Let us recall that a clique is a subset X of V such that all pairs of nodes in X are connected by an edge. The function to be maximized is the clique size $f(X) = |X|$, X being the current clique, and the neighborhood $M(X)$ consists of all cliques that can be obtained from X by adding or dropping a single vertex (*add* or *drop* moves).

At a given iteration, the neighborhood set $M(X)$ is partitioned into the set of *prohibited* neighbors and the set of *allowed* ones. As soon as a vertex is moved (added or removed from the current clique), changing its status is prohibited for the next T iterations; it is allowed otherwise. With a slight abuse of terminology, the terms *allowed* and *prohibited* shall also be applied to vertices.

The top-level description of RLS is shown in Figure 21.5. First (lines 2–5) the relevant variables and structures are initialized: they are the iteration counter t, the prohibition period T, the time t_T of the last

Global variables and data structures	
t	Time (iteration counter)
t_T	Time of last period change
S	Nodes in $V \backslash X$ adjacent to all nodes in X
deltaS[j]	Nodes in $V \backslash S$ adjacent to all nodes in $X \setminus \{j\}$
k_b	Cardinality of best configuration
lastMoved[v]	Time of last movement concerning node v

Local variables	
T	Prohibition period
t_R	Time of last restart
X	Current configuration
I_b	Best configuration
t_b	Time of best configuration

```
 1. procedure Reactive-Local-Search
 2.   $t \leftarrow 0$ ; $T \leftarrow 1$ ; $t_T \leftarrow 0$ ; $t_R \leftarrow 0$
 3.   $X \leftarrow \emptyset$ ; $I_b \leftarrow \emptyset$ ; $k_b \leftarrow 0$ ; $t_b \leftarrow 0$
 4.   S $\leftarrow$ V ; $\forall j \in V$ deltaS[j] $\leftarrow \emptyset$
 5.   $\forall_j \in V$ lastMoved[j] $\leftarrow -\infty$
 6.   repeat
 7.       $T \leftarrow$ History-Reaction ($X$, $T$)
 8.       $X \leftarrow$ Best-Neighbor ($X$)
 9.       $t \leftarrow t + 1$
10.       if $|X| > k_b$
11.           $I_b \leftarrow X$ ; $k_b \leftarrow |X|$ ; $t_b \leftarrow t$
12.       if $t$ - max $\{t_b, t_R\}$ > 100 $k_b$
13.           $t_R \leftarrow t$ ; Restart
14.   until $k_b$ is acceptable
15.       or maximum number of iterations reached
```

FIGURE 21.5 RLS algorithm: pseudocode description.

change of T, the last restart time t_R, the current clique X, the largest clique I_b found so far with its size k_b, and the iteration t_b at which it is found. The set S shall be used by function BEST-NEIGHBOR and contains the set of eligible nodes to improve the current clique (initially, the current clique is empty, and no node is prohibited, so all nodes in V are eligible); the role of array *deltaS* shall be explained in Section 21.4.1.1. Then the loop (lines 6–15) continues to be executed until a satisfactory solution is found or a limiting number of iterations is reached.

The function HISTORY-REACTION searches for the current clique X in memory, inserts it if it is a new one, and adjusts the prohibition T through feedback from the previous history of the search.

Then the best neighbor is selected and the current clique updated (line 8). The iteration counter is incremented. If a better solution is found, the new solution, its size, and the time of the last improvement are saved (lines 10–11). A restart is activated after a suitable number of iterations are executed from the last improvement and from the last restart (lines 12–13).

The prohibition period T is equal to one at the beginning, because in this manner one avoids coming back to the just abandoned clique. Nonetheless, let us note that RLS behaves exactly as local search in the first phase, as long as only new vertices are added to the current clique X, and therefore prohibitions do not have any effect. The difference starts when a maximal clique with respect to set inclusion is reached and the first vertex is dropped.

21.4.1.1 Choice of the Best Neighbor

The function BEST-NEIGHBOR is described in Figure 21.6. Given a current clique X, let us define S as the vertex set of possible additions, i.e., the vertices that are adjacent to all nodes of X. Let $G(S)$ be the subgraph induced by S. Finally, if $j \in X$, *deltaS*$[j]$ is the number of vertices adjacent to all nodes of X but j. A vertex v is *prohibited* at iteration t iff it satisfies *lastMoved*$[v] \geq (t - T^{(t)})$, where *lastMoved*$[v]$ is the last iteration at which it has been added to or dropped from the current clique. Vector *lastMoved* is used to determine the allowed vertices (see lines 3 and 11).

The best neighbor is chosen in stages with this overall scheme: first an allowed vertex that can be added to the current clique is searched for (lines 3–7). If none is found, an allowed vertex to drop is searched for (lines 10–13). Finally, if no allowed moves are available, a random vertex in X is dropped if X is not empty (line 15), a random vertex in V is added in the opposite case (lines 16–18).

Parameters	
X	Configuration to be changed

Local variables	
type	Type of move (addMove or dropMove)
v	Node to be added or removed

```
1.  function BEST-NEIGHBOR ($X$)
2.      $type \leftarrow$ notFound
3.      if {allowed nodes $\in S\} \neq \emptyset$
4.          $type \leftarrow$ addMove
5.          $maxDegAllowed \leftarrow$ maximum degree in $G(S)$
6.          $v \leftarrow$ random allowed $w \in S$
7.              with $\deg_{G(S)}(w) = maxDegAllowed$
8.      if $type =$ notFound and $X \neq \emptyset$
9.          $type \leftarrow$ dropMove
10.         if {allowed $v \in X\} \neq \emptyset$
11.             $maxDeltaS \leftarrow \max_{\text{allowed} j \in X} deltaS[j]$
12.             $v \leftarrow$ random allowed $w \in X$
13.                 with $deltaS[w] = maxDeltaS$
14.         else
15.             $v \leftarrow$ random $w \in X$
16.     if $type =$ notFound
17.         $type \leftarrow$ addMove
18.         $v \leftarrow$ random $w \in V$
19.     INCREMENTAL-UPDATE ($v$, $type$)
20.     if $type =$ addMove return $X \cup \{v\}$
21.         else return $X \setminus \{v\}$
```

FIGURE 21.6 RLS algorithm: the function BEST-NEIGHBOR.

Local variables	
Z	Hash item (or pointer)
$lastVisit[Z]$	Time of last visit of item Z
R	Time since last visit

```
1.  function HISTORY-REACTION ($X$, $T$)
2.      $Z \leftarrow$ HASH-SEARCH ($X$)
3.      if $Z \neq$ null
4.          $R \leftarrow t$ - $lastVisit[Z]$
5.          $lastVisit[Z] \leftarrow t$
6.          if $R < 2(n\text{-}1)$
7.              $t_T \leftarrow t$
8.              return INCREASE($T$)
9.      else
10.         HASH-INSERT($X$, $t$)
11.     if $t$ - $t_T > 10\ k_b$
12.         $t_T \leftarrow t$
13.         return DECREASE($T$)
14.     return $T$
```

FIGURE 21.7 RLS algorithm: routine HISTORY-REACTION.

Ties among *allowed* vertices that can be added are broken by preferring the ones with the largest degree in $G(S)$ (line 7); a random selection is executed among vertices with equal degree in $G(S)$.

Ties among *allowed* vertices that can be dropped are broken by preferring those causing the largest increase $|S^{(t+1)}| - |S^{(t)}|$ where, $S^{(t)}$ is the set S at iteration t (line 13). Again, a random selection is then executed if this criterion selects more that one winner.

21.4.1.2 Reaction and Periodic Restart

The function HISTORY-REACTION is illustrated in Figure 21.7. The prohibition T is minimal at the beginning ($T = 1$), and is then determined by two competing processes: T increases when the current clique comes back to one already found, it decreases when no cliques are repeated in a suitable period. In detail: the current clique X is searched in memory by utilizing hashing techniques (line 1). If X is found, a reference Z is returned to a data structure containing the last visit time (line 2). If the repetition interval R is sufficiently short, cycles are discouraged by increasing T (lines 5–7). If X is not found, it is stored in memory with the time t when it was encountered (line 9). If T remained constant for a number of iterations greater than $10k_b$, and therefore no clique is repeated during this interval, it is decreased (lines 10–12). Increases and decreases, with a minimal change of one unit plus upper and lower bounds, are realized by the two following functions:

$$\text{INCREASE}(T) = \min\{\max\{T \cdot 1.1,\ T + 1\},\ n - 2\}$$
$$\text{DECREASE}(T) = \max\{\min\{T \cdot 0.9,\ T - 1\},\ 1\}$$

The routine RESTART is similar to that in Ref. [57]. If there are vertices that have never been part of the current clique during the search, i.e., that have never been moved since the beginning of the run, one of them with maximal degree in V is randomly selected (lines 3–5 in Figure 21.8). If all vertices have already

Local variables	
L	Nodes in V not moved yet

```
1.  procedure RESTART
2.      $T \leftarrow 1$ ; $t_T \leftarrow t$
3.      if $\exists v \in V(lastMoved[v] = -\infty)$
4.          $L \leftarrow \{w \in V\text{: } lastMoved[w] = -\infty\}$
5.          $v \leftarrow$ random vertex with maximum $\deg_{G(V)}(v)$ in $L$
6.      else
7.          $v \leftarrow$ random vertex $\in V$
8.      relevant data structures are reinitialized
9.      INCREMENTAL-UPDATE ($v$, addMove)
10.     X $\leftarrow \{v\}$
```

FIGURE 21.8 RLS algorithm: routine RESTART.

been members of X in the past, a random vertex in V is selected (line 7). Data structures are updated to reflect the situation of $X = \emptyset$, then the incremental update is applied and the vertex v is added.

21.4.2 Complexity per Iteration

The computational complexity of each iteration of RLS is the sum of a term caused by the usage and updating of reaction-related structures and a term caused by the local search part: neighborhood evaluation and generation of the next clique.

Let us first consider the reaction-related part. The overhead per iteration incurred to determine the prohibitions is $O(|M(X)|)$, $M(X)$ being the neighborhood, that for updating the last usage time of the chosen move is $O(1)$, that to check for repetitions and to update and store the new *hashing value* of the current configuration has an average complexity of $O(1)$, if an incremental *hashing* calculation is applied.

In our case the single-iteration RLS complexity is dominated by the neighborhood evaluation. This evaluation requires an efficient update of the sets S and SMINUS plus the computation of the degrees of the vertices in the induced subgraph $G(S)$ (used in function BEST-NEIGHBOR, Figure 21.6, line 6). It is therefore crucial to consider incremental algorithms, in an effort to reduce the complexity. In our algorithm, the sets S and SMINUS are maintained by the routine INCREMENTAL-UPDATE that is used in the function BEST-NEIGHBOR and in the procedure RESTART. The limited space of this extended abstract force us to omit the detailed description of INCREMENTAL-UPDATE and of the related data structures. The following theorem is proved in the full paper:

Theorem 21.1

The incremental algorithm for updating X, S, and SMINUS *during each iteration of RLS has a worst-case complexity of $O(n)$. In particular, if vertex v is added to or deleted from S, the required operations are $O(\deg_{\overline{G}}(v))$.*

Let us note that the actual multiplicative constant is very small and that the algorithm tends to be faster for dense graphs, where the average degree $\deg_{\overline{G}}(v)$ in the complement graph can be much smaller than n.

Finally, the computation of the vertex degrees in the induced subgraph $G(S)$ costs at most $O(m)$ by the following trivial algorithm. All the edges are inspected, if both endpoints are in S, the corresponding degrees are incremented by 1. In practice the degree is not computed from scratch but it is updated incrementally with a much lesser computational effort; in fact, the maximum number of nodes that enter or leave S at a given iteration is at most $\deg_{\overline{G}}(v)$, v being the just moved vertex. Therefore, the number of operations performed is at most $O(\deg_{\overline{G}}(v) \cdot |S^{(t+1)}|)$. Because the search aims at maximizing the clique X, the set S tends to be very small (at some steps empty!) after a first transient period, and the dominant factor is the same $O(\deg_{\overline{G}}(v))$ factor that appears in the above theorem.

Putting together all the complexity considerations the following corollary is immediately implied:

Corollary 21.1

The worst-case complexity of a single iteration is $O(\max\{n, m\})$.

Experimental results of RLS on the benchmark suite for the MC problem by the organizers of the Second DIMACS Implementation Challenge [55] are analyzed in Ref. [33].

21.5 Related Reactive Approaches

Because Reactive Search is rooted in the automation of the algorithm tuning process by embodying the typical experimental cycle followed by heuristic algorithm designers, it is not surprising to see related ideas and principles arising in various areas. For space limitation we list in this section a limited selection of interesting related papers.

Probabilistic methods that explicitly maintain statistics about the search space by creating models of the good solutions found so far are considered, for example, in Refs. [58,59].

Implicit models of the configuration space are built by a population of searchers in Genetic Algorithms, where learning comes in the form of "survival of the fittest" and generation of new sampling points depends on the previous evolution (see, e.g., Ref. [60]). The issue of controlling parameters of an evolutionary algorithm, including adaptive and "self-adaptive" techniques is considered in Ref. [61], while fitness landscape analysis for the choice of appropriate operators in "memetic" algorithms (combining genetic algorithms with local search) is considered in Ref. [62].

In the area of stochastic optimization, which considers noise in the evaluation process, memory-based schemes for validation and tuning of function approximators are used in Refs. [63,64].

Guided local search aims at exploiting the problem and search-related information to effectively guide local search heuristics [65]. Evaluation functions for global optimization and Boolean satisfiability are learnt in Ref. [66].

Learning mechanisms with biological motivations are used in Ant Colony Optimization based on feedback, distributed computation, and the use of a constructive greedy heuristic. For example, "pheromone trail" information to perform modifications on solutions for the quadratic assignment problem is considered in Refs. [67,68].

Dynamic local search, which increases penalties of some solution components to move the tentative solution away from a given local minimum can also be considered as a form of learning based on the previous history of the search. An example is the dynamic local search algorithm on the MAXSAT problem in Ref. [69], see also Ref. [70] for additional references about stochastic local search methods including adaptive versions.

In the area of computer systems management, "autonomic" systems are based on self-regulating biological systems (http://www.research.ibm.com/autonomic/), which have many points of contact with Reactive Search.

The authors wish to acknowledge support by the BIONETS Project (IST-027748) funded by the FET Program of the European Commission.

References

[1] Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C., and Stewart, W., Designing and reporting on computational experiments with heuristic methods, *J. Heuristics,* 1(1), 9, 1995.

[2] Hooker, J. N., Testing heuristics: we have it all wrong, *J. Heuristics,* 1(1), 33, 1995.

[3] McGeoch, C. C., Toward an experimental method for algorithm simulation, *INFORMS J. Comput.,* 8(1), 1, 1996.

[4] Valiant, L. G., A theory of the learnable, *CACM,* 27(11), 1134, 1984.

[5] Kearns, M. J. and Vazirani, U. V., *An Introduction to Computational Learning Theory,* The MIT Press, Cambridge, MA, 1994.

[6] Kirkpatrick, S., Jr., Gelatt, C. D., and Vecchi, M. P., Optimization by simulated annealing, *Science,* 220, 671, 1983.

[7] Battiti, R. and Tecchiolli, G., Simulated annealing and tabu search in the long run: a comparison on QAP tasks, *Comput. Math. Appl.,* 28(6), 1, 1994. Most of the Battiti's papers are available in postscript form at http://rtm.science.unitn.it/~battiti/battiti-publications.html.

[8] Battiti, R. and Tecchiolli, G., Local search with memory: benchmarking RTS, *Oper. Res. Spektrum,* 17(2/3), 67, 1995.

[9] Aarts, E. H. L., Korst, J. H. M., and Zwietering, P. J., Deterministic and randomized local search, in *Mathematical Perspectives on Neural Networks,* Mozer, M., Smolensky, P., and Rumelhart, D., Eds., Lawrence Erlbaum Publishers, Hillsdale, NJ, 1995.

[10] Ferreira, A. G. and Zerovnik, J., Bounding the probability of success of stochastic methods for global optimization, *Comput. Math. Appl.,* 25, 1, 1993.

[11] Chiang, T. S. and Chow, Y., On the convergence rate of annealing processes, *SIAM J. Control and Optimization,* 26(6), 1455, 1988.

[12] Strenski, P. N. and Kirkpatrick, S., Analysis of finite length annealing schedules, *Algorithmica,* 6, 346, 1991.

[13] Glover, F., Tabu search—part I, *ORSA J. Comput.,* 1(3), 190, 1989.

[14] Steiglitz, K. and Weiner, P., Algorithms for computer solution of the traveling salesman problem, in *Proc. 6th Allerton Conf. on Circuit and System Theory,* Urbana, Illinois, 1968, p. 814.

[15] Lin, S., Computer solutions of the travelling salesman problems, *Bell Syst. Tech. J.,* 44(10), 2245, 1965.

[16] Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization, Algorithms and Complexity*, Prentice-Hall, New Jersey, 1982.

[17] Glover, F., Tabu search—part II, *ORSA J. Comput.,* 2(1), 4, 1990.

[18] Hansen, P. and Jaumard, B., Algorithms for the maximum satisfiability problem, *Computing,* 44, 279, 1990.

[19] Glover, F., Tabu search: improved solution alternatives, in *Mathematical Programming, State of the Art,* Birge, J. R. and Murty, K. G., Eds., The University of Michigan, Michigan, 1994, p. 64.

[20] Battiti, R. and Tecchiolli, G., The reactive tabu search, *ORSA J. Comput.,* 6(2), 126, 1994.

[21] Woodruff, D. L. and Zemel, E., Hashing vectors for tabu search, *Ann. Oper. Res.,* 41, 123, 1993.

[22] Battiti, R., Time- and Space-Efficient Data Structures for History-Based Heuristics, Technical report UTM-96-478, Dip. di Matematica, Univ. di Trento, 1996.

[23] Battiti, R., Partially persistent dynamic sets for history-sensitive heuristics, in *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Challenges,* Johnson, D. S., Goldwasser, M. H., and McGeoch, C. C., Eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 59, AMS, Providence, RI, 2002, p. 1.

[24] Driscoll, J. R., Sarnak, N., Sleator, D. D., and Tarjan, R. E., Making data structures persistent, in *Proc. of STOC,* 1986.

[25] Overmars, M. H., Searching in the Past II: General Transforms, Technical report, Department of Computer Science, University of Utrecht, The Netherlands, 1981.

[26] Bayer, R., Symmetric binary B-trees: data structure and maintenance algorithms, *Acta Inform.,* 1, 290, 1972.

[27] Guibas, L. J. and Sedgewick, R., A dichromatic framework for balanced trees, in *Proc. of FOCS,* 1978, p. 8.

[28] Tarjan, R. E., Updating a balanced search tree in $O(1)$ rotations, *Inf. Process. Lett.,* 16, 253, 1983.

[29] Maier, D., and Salveter, S. C., Hysterical B-trees, *Inf. Process. Lett.,* 12(4), 199, 1981.

[30] Dammeyer, F. and Voss, S., Dynamic tabu list management using the reverse elimination method, *Ann. Oper. Res.,* 41, 31, 1993.

[31] Sarnak, N. and Tarjan, R. E., Planar point location using persistent search trees, *CACM,* 29(7), 669, 1986.

[32] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C., Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning, *Oper. Res.,* 39(3), 378, 1991.

[33] Battiti, R. and Protasi, M., Reactive local search for the maximum clique problem, *Algorithmica,* 29(4), 610, 2001.

[34] Fox, B. L., Simulated annealing: folklore, facts, and directions, in *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing,* Niederreiter, H. and Shiue, P. J.-S., Eds., Springer, Berlin, 1995.

[35] Battiti, R. and Tecchiolli, G., Training neural nets with the reactive tabu search, *IEEE Trans. Neural Networks,* 6(5), 1185, 1995.

[36] Chiang, W. C. and Russell, R. A., A reactive tabu search metaheuristic for the vehicle routing problem with time windows, *INFORMS J. Comput.,* 9, 417, 1997.

[37] Osman, I. H. and Wassan, N. A., A reactive tabu search meta-heuristic for the vehicle routing problem with back-hauls, *J. Scheduling,* 5(4), 287, 2002.

[38] Bräysy, O., A reactive variable neighborhood search for the vehicle-routing problem with time windows, *INFORMS J. Comput.,* 15(4), 347, 2003.

[39] Kincaid, R. K. and Labal, K. E., Reactive tabu search and sensor selection in active structural acoustic control problems, *J. Heuristics,* 4(3), 199, 1998.

[40] Anzellotti, G., Battiti, R., Lazzizzera, I., Soncini, G., Zorat, A., Sartori, A., Tecchiolli, G., and Lee, P., TOTEM: a highly parallel chip for triggering applications with inductive learning based on the reactive tabu search, *Int. J. Mod. Phys. C,* 6(4), 555, 1995.

[41] Battiti, R. and Bertossi, A. A., Greedy, prohibition, and reactive heuristics for graph partitioning, *IEEE Trans. Comput.,* 48(4), 361, 1999.

[42] Toune, S., Fudo, H., Genji, T., Fukuyama, Y., and Nakanishi, Y., Comparative study of modern heuristic algorithms to service restoration in distribution systems, *IEEE Trans. Power Delivery,* 17(1), 173, 2002.

[43] Battiti, R. and Protasi, M., Reactive search, a history-sensitive heuristic for MAX-SAT, *ACM J. Exp. Algorithmics,* 2, 1997.

[44] Battiti, R. and Protasi, M., Reactive local search techniques for the maximum k-conjunctive constraint satisfaction problem, *Discrete Appl. Math.,* 96–97, 3, 1999.

[45] Nonobe, K. and Ibaraki, T., A tabu search approach for the constraint satisfaction problem as a general problem solver, *Eur. J. Oper. Res.,* 106, 599, 1998.

[46] Battiti, R. and Tecchiolli, G., The continuous reactive tabu search: blending combinatorial optimization and stochastic search for global optimization, *Ann. Oper. Res.—Metaheuristics Comb. Optimization,* 63, 153, 1996.

[47] Chelouah, R. and Siarry, P., Tabu search applied to global optimization, *Eur. J. Oper. Res.,* 123, 256, 2000.

[48] Battiti, R. and Brunato, M., Reactive search for traffic grooming in WDM networks, in *Evolutionary Trends of the Internet, IWDC2001,* Palazzo, S., Ed., Lecture Notes in Computer Science, Vol. 2170, Springer, Berlin, 2001, p. 56.

[49] Winter, T. and Zimmermann, U., Real-time dispatch of trams in storage yards, *Ann. Oper. Res.,* 96, 287, 2000.

[50] Fortz, B. and Thorup, M., Increasing Internet capacity using local search, *Comput. Optimization Appl.,* 29(1), 13, 2004.

[51] Ausiello, G., Crescenzi, P., and Protasi, M., Approximate solution of NP optimization problems, *Theor. Comput. Sci.,* 150, 1, 1995.

[52] Crescenzi, P. and Kann, V., A Compendium of NP Optimization Problems, Technical report, 1996, electronic notes: http://www.nada.kth.se/˜viggo/problemlist/compendium.html.

[53] Bellare, M., Goldreich, O., and Sudan, M., Free bits, PCP and non-approximability—towards tight results, *Proc. of FOCS,* 1995, p. 422.

[54] Hastad, J., Clique is hard to approximate within $n^{1-\epsilon}$, *Proc. of FOCS,* 1996, p. 627.

[55] Johnson, D. S. and Trick, M., Eds., *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge,* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26, AMS, Providence, RI, 1996.

[56] Pardalos, P. M. and Xu, J., The maximum clique problem, *J. Global Optimization,* 4, 301, 1994.

[57] Soriano, P. and Gendreau, M., Tabu Search Algorithms for the Maximum Clique Problem, Technical report CRT-968, Centre de Recherche sur les Transports, Universite de Montreal, Canada, 1994.

[58] Pelikan, M., Goldberg, D. E., and Lobo, F., A survey of optimization by building and using probabilistic models, *Comput. Optimization Appl.,* 21(1), 5, 2002.

[59] Baluja, S. and Davies, S., Using optimal dependency trees for combinatorial optimization: learning the structure of the search space, in *Proc. 14th Int. Conf. Machine Learning,* Fisher, D. H., Ed., Morgan Kaufmann, San Mateo, CA, 1997, p. 30.

[60] Syswerda, G., Simulated crossover in genetic algorithms, in *Foundations of Genetic Algorithms,* Whitley, D. L., Ed., Morgan Kaufmann, San Mateo, CA, 1993, p. 239.

[61] Eiben, A. E., Hinterding, R., and Michalewicz, Z., Parameter control in evolutionary algorithms, *IEEE Trans. Evol. Comput.,* 3(2), 124, 1999.

[62] Merz, P. and Freisleben, B., Fitness landscape analysis and memetic algorithms for the quadratic assignment problem, *IEEE Trans. Evol. Comput.,* 4(4), 337, 2000.

[63] Moore, A. W. and Schneider, J., Memory-based stochastic optimization, in *Advances in Neural Information Processing Systems,* Vol. 8, Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., Eds., The MIT Press, Cambridge, MA, 1996, p. 1066.

[64] Dubrawski, A. and Schneider, J., Memory based stochastic optimization for validation and tuning of function approximators, *Conference on AI and Statistics,* 1997.

[65] Voudouris, Ch. and Tsang, E., Guided local search and its application to the traveling salesman problem, *Eur. J. Oper. Res.,* 113, 469, 1999.

[66] Boyan, J. A. and Moore, A. W., Learning evaluation functions for global optimization and boolean satisfiability, *Proc. 15th National Conf. on Artificial Intelligence (AAAI),* AAAI Press, Menlo Park, CA, USA, 1998, p. 3.

[67] Dorigo, M., Maniezzo, V., and Colorni, A., Ant system: optimization by a colony of cooperating agents, *IEEE Trans. Syst., Man Cybernetics, Part B,* 26(1), 29, 1996.

[68] Gambardella, L. M., Taillard, E. D., and Dorigo, M., Ant colonies for the quadratic assignment problem, *J. Oper. Res. Soc.,* 50(2), 167, 1999.

[69] Hutter, F., Tompkins, D. A. D., and Hoos, H. H., Scaling and probabilistic smoothing: efficient dynamic local search for SAT, in *Proc. of CP-02,* Lecture Notes on Computer Science, Vol. 2470, Springer, Berlin, 2002, p. 233.

[70] Hoos, H. H. and Stuetzle, T., *Stochastic Local Search: Foundations and Applications,* Morgan Kaufmann, San Mateo, CA, 2005.

22

Neural Networks

Bhaskar DasGupta[*]
University of Illinois at Chicago

Derong Liu
University of Illinois at Chicago

Hava T. Siegelmann
University of Massachusetts

22.1 Introduction

Artificial neural networks have been proposed as a tool for machine learning (see, e.g., Refs. [1–4]) and many results have been obtained regarding their application to practical problems in robotics control, vision, pattern recognition, grammatical inferences, and other areas (see, e.g., Refs. [5–8]). In these roles, a neural network is trained to recognize complex associations between inputs and outputs that were presented during a *supervised* training cycle. These associations are incorporated into the weights of the network, which encode a distributed representation of the information that were contained in the input patterns. Once trained, the network will compute an input/output mapping which, if the training data were representative enough, will closely match the unknown rule which produced the original data. Massive parallelism of computation, as well as noise and fault tolerance, are often offered as justifications for the use of neural nets as learning paradigms.

Traditionally, especially in the structural complexity literature (see, e.g., Ref. [4]), feedforward circuits composed of AND, OR, NOT, or threshold gates have been thoroughly studied. However, in practice, when designing a neural net, continuous activation functions such as the *standard sigmoid* are more commonly used. This is because usual learning algorithms such as the backpropagation algorithm assumes a continuous activation function. As a result, neural nets are distinguished from those conventional circuits because they perform real-valued computation and admit efficient learning procedures. The last three decades have seen a resurgence of theoretical techniques to design and analyze the performances of neural nets (see, e.g., the survey in Ref. [9]) as well as novel application of neural nets to various applied areas (see, e.g., Ref. [6] and some of the references there). Theoretical researches in computational capabilities of neural nets have given valuable insights into the mechanisms of these models.

In subsequent discussions, we distinguish between two types of neural networks, commonly known as the "feedforward" neural nets and the "recurrent" neural nets. A feedforward net consists of a number of processors ("nodes" or "neurons") each of which computes a function of the type $y = \sigma(\sum_{i=1}^{k} a_i u_i + b)$ of its inputs $u_1, \ldots, u_k$. These inputs are either external (input data are fed through them) or they represent the outputs y of other nodes. No cycles are allowed in the connection graph and the output of one designated node is understood to provide the output value produced by the entire network for a given

[*] B. DasGupta was supported in part by NSF grants CCR-0206795, CCR-0208749, and IIS-0346973.

vector of input values. The possible coefficients a_i and b appearing in the different nodes are the *weights* of the network, and the functions σ appearing in the various nodes are the *node*, *activation*, or *gate* functions. An *architecture* specifies the interconnection structure and the σs, but not the actual numerical values of the weights. A recurrent neural net, however, allows cycles in the connection graph, thereby allowing the model to have substantially more computational capabilities (see Section 22.3.2).

In this chapter we survey research works dealing with basic questions regarding *computational capabilities* and *learning* of neural models. There are various types of such questions that one may ask, most of them closely related and complementary to each other. We next describe a few of them *informally*.

One direction of research deals with the *representational capabilities* of neural nets, assuming *unlimited* number of neurons are available (see, e.g., Refs. [10–16]). The origin of this type of research can be traced back to the work of the famous mathematician Kolmogorov [17], who essentially proved the first existential result on the representation capabilities of depth 2 neural nets. This type of research ignores the training question itself, asking instead if it is *at all* possible to compute or approximate arbitrary functions (e.g., Refs. [11–17]) or if the net can simulate, say, Turing machines (e.g., Refs. [15,16]). Many of the results and proofs in this direction are nonconstructive.

Another perspective to learnability questions of neural nets takes a *numerical analysis or approximation theoretic* point of view. There one asks questions such as *how many* hidden units are necessary to well approximate, that is to say, approximate with a small overall error, an unknown function. This type of research also ignores the training question, asking instead what is the best one could do, in this sense of overall error, if the best possible network with a given architecture were to be eventually found. Some papers along these lines are Refs. [18,19], which dealt with single hidden layer nets and Ref. [20], which dealt with multiple hidden layers.

Another possible line of research deals with the *sample complexity* questions, that is, the quantification of the amount of information (number of samples) needed to characterize a given unknown mapping. Some recent references to such work, establishing sample complexity results, and hence "weak learnability" in the Valiant model, for neural nets, are the papers in Refs. [21–25]; the first of these references deals with networks that employ hard threshold activations, the third and fourth cover continuous activation functions of a type (piecewise polynomial), and the last one provides results for networks employing the standard sigmoid activation function.

Yet another direction in which to approach theoretical questions regarding learning by neural networks originates with the work of Judd (see, e.g., Refs. [26,27] as well as the related work [28,29]). Judd was motivated by the observation that the "backpropagation" algorithm often runs very slowly, especially for high-dimensional data. Recall that this algorithm is used to find a network (i.e., find the weights, assuming a fixed architecture) that reproduces the observed data. Of course, many modifications of the vanilla "backprop" approach are possible, using more sophisticated techniques such as high-order (Newton), conjugate gradient, or sequential quadratic programming methods. However, the "curse of dimensionality" seems to arise as a computational obstruction to all these training techniques as well, when attempting to learn arbitrary data using a standard feedforward network. For the simpler case of linearly separable data, the perceptron algorithm and linear programming techniques help to find a network—with no "hidden units"—relatively fast. Thus one may ask if there exists a *fundamental barrier* to training by general feedforward networks, a barrier that is insurmountable no matter which particular algorithm one uses. (Those techniques which *adapt* the architecture to the data, such as cascade correlation or incremental techniques, would not be subject to such a barrier.)

22.2 Feedforward Neural Networks

As mentioned earlier, a feedforward neural net is one in which the underlying connection graph contains *no* directed cycles. More precisely, a feedforward neural net (or, in our terminology, a Γ net) can be defined as follows.

Let Γ be a class of real-valued functions, where each function is defined on some subset of $\mathbb{R}$. A Γ net C is an unbounded fan-in circuit whose edges and vertices are labeled by real numbers. The real number

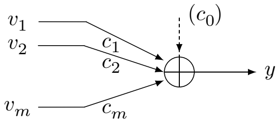


FIGURE 22.1 Classical perceptrons.

assigned to an edge (resp. vertex) is called its *weight* (resp. its *threshold*). Moreover, to each vertex v a *gate* (or *activation)* function $\gamma_v \in \Gamma$ is assigned.

The circuit C *computes* a function $f_C : \mathbb{R}^m \to \mathbb{R}$ as follows. The components of the input vector $x = (x_1, \ldots, x_m) \in \mathbb{R}^m$ are assigned to the sources of C. Let $v_1, \ldots, v_n$ be the immediate predecessors of a vertex v. The input for v is then $s_v(x) = \sum_{i=1}^{n} w_i y_i - t_v$, where w_i is the weight of the edge (v_i, v), t_v the *threshold* of v, and y_i the value assigned to v_i. If v is not a sink, then we assign the value $\gamma_v(s_v(x))$ to v. Otherwise we assign $s_v(x)$ to v.

Without any loss of generality, one can assume that C has a single sink t. Then $f_C = s_t$ is the function computed by C.

The function class Γ, quite popular in the structural-complexity literature (see, e.g., Refs. [30–32]), is the *binary threshold function* $\mathcal{H}$ defined by

$$\mathcal{H}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

However, in practice this function is not so popular because the function is discrete, and hence using this gate function may pose problems in most commonly used learning algorithms like the backpropagation algorithms [3] or their variants. Also, from biological perspectives, real neurons have continuous input–output relations [33]. In practice, various continuous (or, at least locally smooth) gate functions have been used, for example, *the cosine squasher, the standard sigmoid, radial basis functions, generalized radial basis functions, piecewise linear, polynomials, and trigonometric polynomial* functions. In particular, the standard sigmoid function $\sigma(x) = 1/(1 + \mathbf{e}^{-x})$ is very popular.

The simplest type of feedforward neural net is the classical *perceptron.* This consists of a single neuron computing a threshold function (see Figure 22.1). In other words, the perceptron P is characterized by a vector (of "weights") $\vec{c} \in \mathbb{R}^m$, and computes the inner product $\vec{c} \cdot v + c_0 = c_1 v_1 + \cdots + c_m v_m + c_0$. Such a model has been well studied and efficient learning algorithms for it exists (e.g., Ref. [34], see also Ref. [35]). In this chapter we will be, however, more interested in more complex *multilayered neural nets* (see Figure 22.2).

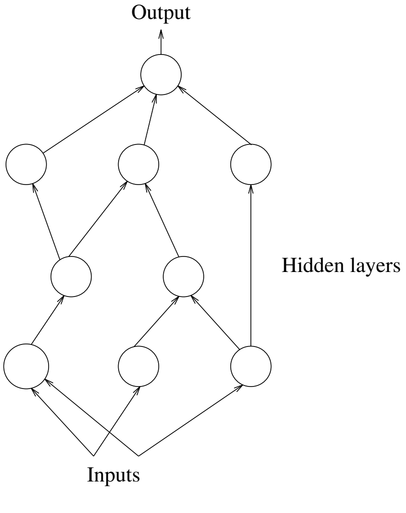


FIGURE 22.2 A feedforward neural net with three hidden layers and two inputs.

22.2.1 Approximation Properties

It is known that neural nets of only depth 2 and with arbitrarily large number of nodes can approximate any real-valued function up to any desired accuracy, using a continuous activation function such as the sigmoidal function (see e.g., Refs. [11,17]). However, these proofs are mostly nonconstructive and, from a practical point of view, one is more interested in designing *efficient* neural nets (i.e., roughly speaking, neural nets with small size and depth) to exactly or approximately compute different functions. It is also important, from a practical point of view, to understand the *size–depth trade-off* of the complexity of feedforward nets while computing functions, since generally neural nets with more layers are more costly to simulate or implement.

Threshold circuits, i.e., feedforward nets with threshold activation functions, have been quite well studied, and upper/lower bounds for them have been obtained while computing various Boolean functions (see, e.g., Refs. [30–32,36–38] among many other works). Functions of special interest have been the parity function, computing the multiplication and division of binary numbers, and so forth.

However, as mentioned earlier, it is more common in practice to use a continuous activation function, such as the standard sigmoid function. Refs. [39,40], among others, considered efficient computation or approximation of various functions by feedforward circuits with continuous activation functions and also studied size–depth trade-offs. In particular, Ref. [39] showed that any polynomial of degree n with polynomially bounded coefficients can be approximated with exponential accuracy by depth 2 feedforward sigmoidal neural nets with a polynomial number of nodes. Refs. [39,40] also show how to simulate threshold circuits by sigmoidal circuits with a polynomial increase in size and a constant factor increase in depth. Thus, in effect, functions computed by threshold circuits can also be computed by sigmoidal circuits with not too much increase in size and depth. Maass [23] shows how to simulate nets with piecewise-linear activation functions with bounded depth, arbitrary real weights, and for Boolean inputs and outputs by a threshold net of somewhat larger size and depth with weights from $\{-1, 0, 1\}$. Refs. [39,40] showed that circuits composed of sufficiently smooth gate functions are capable of efficiently approximating polynomials within any degree of accuracy. Complementing these results, Ref. [39] also provided nontrivial lower bounds on the size of *bounded-depth* sigmoidal nets with polynomially large weights when computing oscillatory functions. In essence, one can prove results of the following types.

Definition 22.1 **(DasGupta and Schnitger [39])**[1]

Let $\gamma : \mathbb{R} \to \mathbb{R}$ be a function. We call γ nontrivially smooth with parameter k if and only if there exists rational numbers α, $\beta (\alpha > 0)$, and an integer k such that α and β have logarithmic size at most k and

- *(a) γ can be represented by the power series $\sum_{i=0}^{\infty} a_i (x-\beta)^i$ for all $x \in [\beta-\alpha, \beta+\alpha]$. For each $i > 1$, a_i is a rational number of logarithmic size at most i^k.*
- *(b) For each $i > 1$ there exists j with $i \le j \le i^k$ and $a_j \neq 0$.*
- *(c) For each $i > 1$, $||\gamma^{(i)}||_{[-\alpha,\alpha]} \le 2^{i^k}$.*

Theorem 22.1 **(DasGupta and Schnitger [39])**[2]

Assume that γ is nontrivially smooth with parameter k. Let $p(x)$ be a degree n polynomial whose coefficients are rational numbers of logarithmic size at most max. Then $p(x)$ can be ε approximated (over the domain $[-D, D]$ with $[\beta-\alpha, \beta+\alpha] \subseteq [-D, D]$) by a $\{\gamma\}$-circuit C_p. C_p has depth 2 and size $O(n^{2k})$. The Lipschitz-bound[3] of C_p (over $[-D, D]^n$) is at most $c_\gamma \cdot \left(2^{max} \cdot (2+D) \cdot \frac{1}{\varepsilon}\right)^{poly(n)}$, where the constant c_γ depends only on γ and not on p.

[1] The notation $\gamma^{(i)}$ denotes the ith derivative of γ.

[2] $poly(n)$ denotes a polynomial in n.

[3] The Lipschitz bound of the net is a measure of the numerical stability of the circuit. Informally speaking, a net has a Lipschitz bound of L if all its weights and thresholds are bounded in absolute value by L.

Theorem 22.2 **(DasGupta and Schnitger [39])**

Let $f : [-1, 1] \to \mathbb{R}$ be a function that ε oscillates t times[4] *and let C be a Γ circuit of depth d, size s, and Lipschitz-bound 2^s over $[-1, 1]$. If C approximates f with error at most $\frac{\varepsilon}{4}$, then $s \geq t^{\Omega(1/d)}$.*

Note that the standard sigmoid is nontrivially smooth with a constant k, and so is the case for most of the other continuous activation functions mentioned in Section 22.1. However, the simulation in Theorem 22.1 needs quadratically many nodes to simulate a polynomial by sigmoidal nets with exponential accuracy. Unfortunately, the proof of Theorem 22.2 relies on efficient simulation of a sigmoidal circuit by a spline circuit and hence cannot be extended to the case of arbitrary weights (i.e., the Lipschitz-bound condition cannot be dropped).

22.2.2 Backpropagation Algorithms

Basic backpropagation [41] is currently the most popular supervised learning method that is used to train multilayer feedforward neural networks with differentiable transfer functions. It is a gradient descent algorithm in which the network weights are moved along the negative of the gradient of the performance function.

The basic backpropagation algorithm performs the following steps:

1. *Forward pass.* Inputs are presented and the outputs of each layer are computed.
2. *Backward pass.* Errors between the target and the output are computed. Then, these errors are "backpropagated" from the output to each layer until the first layer. Finally, the weights are adjusted according to the gradient descent algorithm with the derivatives obtained by backpropagation.

We will discuss in more detail a generalized version of this approach for recurrent networks, termed as the "real-time backpropagation through time," (BPTT) in Section 22.3.1. The key point of basic backpropagation is that the weights are adjusted in response to the derivatives of performance function with respect to weights, which only depend on the current pattern; the weights can be adjusted sequentially or in batch mode. More details about the basic backpropagation can be found in Ref. [41]. The asymptotic convergence rates of backpropagation is proved in Ref. [42]. Traditionally, the parity function has been used as an important benchmark for testing the efficiency of a learning algorithm. Empirical studies in Ref. [43] show that the training time of a feedforward net using backpropagation while learning the parity function grows exponentially with the number of inputs, thereby rendering the learning algorithm to be very time consuming. Unfortunately, a satisfactory theoretical justification for this behavior is yet to be shown. Also, it is well known that the backpropagation algorithm may get stuck in local minima, and in fact, in general, gradient descent algorithms may fail to classify correctly data that even simple perceptrons can classify correctly (see, e.g., Refs. [44–46]). Strategies of avoiding local minima include local perturbation and simulated-annealing techniques, whereas the later problem (in absence of a local minima) can be avoided using, say, threshold LMS procedures.

22.2.3 Learning Theoretic Results

Approximation results discussed in Section 22.2.1 do not necessarily translate into good learning algorithms. For example, even though a sigmoidal net has great computational power, we still need to investigate how to learn the weights of such a network from a set of examples. Learning is a very important aspect of designing efficient neural models from a practical point of view. There are a few possible approaches to tackle this issue; we describe one of those approaches next.

[4] f ε oscillates t times if and only if there are real numbers $-1 \leq x_1 < \cdots < x_{t+1} \leq 1$ such that (a) $f(x_1) = f(x_2) = \cdots = f(x_{t+1})$, (b) $|x_{i+1} - x_i| \geq \varepsilon$ for all i, and (c) there are real numbers $y_1, \ldots, y_t$ such that $x_i \leq y_i \leq x_{i+1}$ and $|f(x_i) - f(y_i)| \geq \varepsilon$ for all i.

22.2.3.1 VC-Dimension Approach

Vapnik–Chervonenkis (VC) dimensions (and, their suitable extensions to real-valued computations) provide *information-theoretic bounds* to the sample complexities for learning problems in neural nets. We very briefly (also, somewhat informally) review some (by now standard) notions regarding sample complexity which deals with the calculation of VC dimensions as applicable for neural nets (for more details, see Refs. [9,47–49].

In the general classification problem, an input space $\mathbb{X}$ as well as a collection $\mathcal{F}$ of maps $\mathbb{X} \to \{-1, 1\}$ are assumed to have been given. (The set $\mathbb{X}$ is assumed to be either countable or an Euclidean space, and the maps in $\mathcal{F}$, the set of functions computable by the specific neural nets under consideration, are assumed to be measurable. In addition, mild regularity assumptions are made which ensure that all sets appearing below are measurable, but details are omitted since in the context of neural nets these assumptions are almost always satisfied.) Let W be the set of all sequences

$$w = (u_1, \psi(u_1)), \ldots, (u_s, \psi(u_s))$$

over all $s \geq 1$, $(u_1, \ldots, u_s) \in \mathbb{X}^s$, and $\psi \in \mathcal{F}$. An *identifier* is a map $\varphi : W \to \mathcal{F}$. The value of φ on a sequence w as above will be denoted as φ_w. The *error* of φ with respect to a probability measure P on $\mathbb{X}$, a $\psi \in \mathcal{F}$, and a sequence $(u_1, \ldots, u_s) \in \mathbb{X}^s$, is

$$\text{Err}_\varphi(P, \psi, u_1, \ldots, u_s) := \text{Prob}\,[\varphi_w(u) \neq \psi(u)]$$

(where the probability is being understood with respect to P).

The class $\mathcal{F}$ of functions is said to be (uniformly) *learnable* if there is some identifier φ with the following property: For each $\varepsilon, \delta > 0$ there is some s so that, for every probability P and every $\psi \in \mathcal{F}$,

$$\text{Prob}\,[\text{Err}_\varphi(P, \psi, u_1, \ldots, u_s) > \varepsilon] < \delta$$

(where the probability is being understood with respect to P^s on $\mathbb{X}^s$).

In the learnable case, the function $s(\varepsilon, \delta)$ which provides, for any given ε and δ, the smallest possible s as above, is called the *sample complexity* of the class $\mathcal{F}$. It can be proved that learnability is equivalent to finiteness of a combinatorial quantity called *VC dimension* ν of the class $\mathcal{F}$ in the following sense (cf. Refs. [49,50]):

$$s(\varepsilon, \delta) \;\leq\; \max\left\{ \frac{8\nu}{\varepsilon} \log\left(\frac{13}{\varepsilon}\right), \frac{4}{\varepsilon} \log\left(\frac{2}{\delta}\right) \right\}$$

Moreover, lower bounds on $s(\varepsilon, \delta)$ are also known, in the following sense (cf. Ref. [49]): for $0 < \varepsilon < \frac{1}{2}$, and assuming that the collection $\mathcal{F}$ is not trivial (i.e., $\mathcal{F}$ does not consist of just one mapping or a collection of two disjoint mappings, see Ref. [49] for details), we must have

$$s(\varepsilon, \delta) \;\geq\; \max\left\{ \frac{1-\varepsilon}{\varepsilon} \ln\left(\frac{1}{\delta}\right), \nu(1 - 2(\varepsilon(1-\delta) + \delta)) \right\}$$

The above bounds motivate studies dealing with estimating VC dimensions of neural nets. When there is an algorithm that allows computing an identifier φ in time polynomial on the sample size, the class is said to be learnable in the PAC ("probably approximately correct") sense of Valiant (cf. Ref. [51]). Generalizations to the learning of real-valued (as opposed to Boolean) functions computed by, say, sigmoidal neural nets, by evaluation of the "pseudodimension," are also possible; see the discussion in Ref. [9].

It is well known that a simple perceptron with n inputs has a VC dimension of $n + 1$ [49]. However, the VC dimension of a threshold network with w programmable parameters is $\Theta(w \log w)$ [21,23,52,53]. Maass [23] and Goldberg and Jerrum [24], among others, investigated VC dimensions of neural nets with continuous activations and showed polynomial bounds on neural nets with piecewise-polynomial activation functions. Even finiteness of the VC dimension of sigmoidal neural nets was unknown for a long time, until it was showed to be finite [25]. Subsequently, an $O(w^2 n^2)$ bound on the VC dimension of sigmoidal neural nets, where w is the number of programmable parameters and n the number of nodes, was established [54]. Ref. [55] gives a $\Omega(w^2)$ lower bound for sigmoidal nets.

22.2.3.2 The Loading (Consistency) Problem

The VC dimensions provide information-theoretic bounds on sample complexities for learning. To design an *efficient* learning algorithm, the learner should be able to design a neural net consistent with the (polynomially many) samples it receives. This is known as the *consistency* or the *loading* problem. In other words, now we consider the *tractability* of the training problem, that is, of the question (essentially quoting Judd [27]): "Given a network architecture (interconnection graph as well as choice of activation function) and a set of training examples, does there exist a set of weights so that the network produces the correct output for all examples?"

The simplest neural network, that is, the perceptron, consists of one threshold neuron only. It is easily verified that the computational time of the loading problem in this case is polynomial in the size of the training set irrespective of whether the input takes continuous or discrete values. This can be achieved via a linear programming technique. Blum and Rivest [28] showed that this problem is NP-hard for a simple three-node threshold neural net. Refs. [56,57] extended this result to show NP-hardness of a three-node neural net where the activation function is a simple, saturated piecewise linear activation function, the extension was nontrivial due to the continuous part of the activation function. It was also observed in Ref. [57] that the loading problem is polynomial time if the input dimension is constant. However, the complexity of the loading problem for sigmoidal neural nets still remains an open problem, though some partial results when the net is somewhat restricted appeared in Ref. [58]. Any NP-hardness results of the loading problems also prove hardness in the PAC learning model, due to the result in Ref. [59].

Another possibility to design efficient learning algorithms is to assume that the inputs are drawn according to some particular distributions. For example, see Ref. [60] for efficient learning a depth 2 threshold net with a fixed number of hidden nodes and with the output gate being an AND gate, assuming that the inputs are drawn uniformly from a n-dimensional unit ball.

22.3 Recurrent Neural Networks

As stated in the introduction, a recurrent neural net allows cycles in the connection graph. A sample recurrent neural network is illustrated in Figure 22.3.

22.3.1 Learning Recurrent Networks: Backpropagation through Time

Backpropagation through time is an approach to solve temporal differentiable optimization problems with continuous variables [61] and used most often as a training method for recurrent neural networks. In this section, we describe the method in more detail.

22.3.1.1 Network Definition and Performance Measure

We will use the general expression of Werbos [41] to describe the network dynamics. Symbol y denotes node inputs and outputs, while symbol s denotes the weighted sum of node inputs. An ordered set of i, j, l, k on the weights denotes a connection from node j of layer i to node k of layer l; $w_{0,j,l,k}$ denotes connections from outside the network. The node activation function is denoted by $f(\cdot)$. The last layer of the network is denoted by M. The number of nodes in a layer l is denoted by n_l. The bias inputs to each node are handled homogeneously using the connection weights for zeroth input, where inputs $y_0^{\text{ext}}(t)$ and $y_{l-1,0}(t)$ are fixed at unity for this purpose.

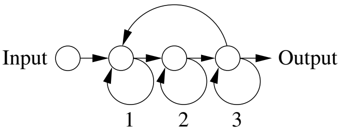


FIGURE 22.3 A simple recurrent network.

All the algorithms presented in the following part of the chapter are based on the following network dynamics expressed in *pseudocode* format:

```
f or k = 1 to n_1{
        $s_{1,k}(t) = \sum_{j=0}^{n^{\text{ext}}} w_{0,j,1,k}(t) y_j^{\text{ext}}(t) + \sum_{j=1}^{n_M} w_{M,j,1,k}(t) y_{M,j}(t-1) + \sum_{j=1}^{n_1} w_{1,j,1,k}(t) y_{1,j}(t-1)$    (22.1)
        $y_{1,k} = f(s_{1,k}(t))$    (22.2)
}

for l = 2 to M{
    for k = 1 to n_l {
        $s_{l,k}(t) = \sum_{j=0}^{n_{l-1}} w_{l-1,j,l,k}(t) y_{l-1,j}(t) + \sum_{j=1}^{n_l} w_{l,j,l,k}(t) y_{l,j}(t-1)$    (22.3)
        $y_{1,k} = f(s_{1,k}(t))$    (22.4)
    }
}
```

Note that the input line is not the first layer. Assume that the task to be performed by the network is sequential supervised learning task, meaning that certain of units' output values are to match specified target values at each time step. Define a time-varying $e_j(t)$:

$$e_j(t) = \begin{cases} d_j(t) - y_j(t) & \text{if } j \in \text{layer } M \\ 0 & \text{otherwise} \end{cases} \tag{22.5}$$

where $d_j(t)$ is the target of the output of the jth unit at time t and define the two performance measure functions:

$$J(t) = \frac{1}{2} \sum_{k \in M} [e_k(t)]^2 \tag{22.6}$$

$$J^{\text{total}}(t_0, t) = \sum_{\tau = t_0}^{t} J(\tau) \tag{22.7}$$

22.3.1.2 Unrolling a Network

In essence, BPTT is the algorithm that calculates derivatives of performance measure with respect to weights for a feedforward neural network, which is obtained by *unrolling* the network in time. Let $\mathcal{N}$ denote the network, which is to be trained to perform a desired sequential behavior. Assume that $\mathcal{N}$ has n units and that it is to run from time t_0 up through some time t. As described by Rumelhart et al. [64], we may "unroll" this network in time to obtain a feedforward network $\mathcal{N}^*$, which has a layer for each time step in $[t_0, t]$ and n units in each layer. Each unit in $\mathcal{N}$ has a copy in each layer of $\mathcal{N}^*$, and each connection from unit j to unit i in $\mathcal{N}$ has a copy connecting unit j in layer τ to unit i in layer $\tau + 1$, for each $\tau \in [t_0, t)$. An example of this unrolling mapping is given in Figure 2 in Ref. [62]. The key value of this conceptualization is that it allows one to regard the problem of training a recurrent network as the corresponding problem of training a feedforward neural network with certain constraints imposed on its weights. The central result driving the BPTT approach is that to compute $\partial J(t_0, t)/\partial w_{ij}$ in $\mathcal{N}$ one simply computes the partial derivatives of $\partial J(t_0, t)$ with respect to each of the τ weights in $\mathcal{N}^*$ corresponding to w_{ij} and adds them up.

Straightforward application of this idea leads to two different algorithms, depending on whether an epochwise or continual operation approach is sought. One is real-time BPTT and the other is epochwise BPTT. We only describe the real-time BPTT because of space limitations.

22.3.1.3 Derivation of BPTT Formulation

Suppose that a differentiable function F expressed in terms of $\{y_{l,j}(\tau)|t_0 \leq \tau \leq t\}$, the outputs of the network over time interval $[t_0, t]$ is given. Note while F may have an *explicit* dependence on $y_{l,j}(\tau)$, it may also have an *implicit* dependence on this same value through later output values. To avoid the ambiguity in interpreting partial derivatives such as $\frac{\partial F}{\partial y_{l,j}(\tau)}$, we introduce variable $y^*_{l,j}(\tau)$ such that $y^*_{l,j}(\tau) = y_{M,j}(\tau)$ for all $l = M$. Define the following:

$$\epsilon_{l,j}(\tau) = \frac{\partial F}{\partial y_{l,j}(\tau)} \tag{22.8}$$

$$\delta_{l,j}(\tau) = \frac{\partial F}{\partial s_{l,j}(\tau)} \tag{22.9}$$

Since F depends on $y_{l,j}(\tau)$, $s_{l,k}(\tau + 1)$, and $s_{l+1,m}(\tau)$, we have

$$\frac{\partial F}{\partial y_{l,j}(\tau)} = \frac{\partial F}{\partial y^*_{l,j}(\tau)} + \sum_{k=1}^{n_l} \frac{\partial F}{\partial s_{l,k}(\tau+1)} \frac{\partial s_{l,k}(\tau+1)}{\partial y_{l,j}(\tau)} + \sum_{m=1}^{n_{l+1}} \frac{\partial F}{\partial s_{l+1,m}(\tau)} \frac{\partial s_{l+1,m}(\tau)}{\partial y_{l,j}(\tau)} \tag{22.10}$$

from which we derive the following:

1. $\tau = t$. For this case,

$$\epsilon_{M,j}(\tau) = \frac{\partial F}{\partial y^*_{M,j}(\tau)} = -e_j(\tau) \tag{22.11}$$

where M means the output layer of the network and $j \in \{1, 2, \ldots, n_M\}$ and

$$\begin{aligned}\epsilon_{l,j}(\tau) = \frac{\partial F}{\partial y_{l,j}(\tau)} &= \sum_{m=1}^{n_{l+1}} \frac{\partial F}{\partial s_{l+1,m}(\tau)} \frac{\partial s_{l+1,m}(\tau)}{\partial y_{l,j}(\tau)} \\ &= \sum_{m=1}^{n_{l+1}} \frac{\partial F}{\partial y_{l+1,m}(\tau)} \frac{\partial s_{l+1,m}(\tau)}{\partial y_{l,j}(\tau)} \frac{\partial s_{l+1,m}(\tau)}{\partial y_{l,j}(\tau)} \\ &= \sum_{m=1}^{n_{l+1}} \epsilon_{l+1,m}(\tau) f^{'}(s_{l+1,m}(\tau)) w_{l,j,l+1,m}\end{aligned} \tag{22.12}$$

where $l = 1, 2, \ldots, M - 1$ and $j = 1, 2, \ldots, n_l$, and

$$\delta_{l,j}(\tau) = \frac{\partial F}{\partial s_{l,j}(\tau)} = \frac{\partial F}{\partial y_{l,j}(\tau)} \frac{\partial y_{l,j}(\tau)}{\partial s_{l,j}(\tau)} = \epsilon_{l,j}(\tau) f^{'}(s_{l,j}(\tau)) \tag{22.13}$$

where $l = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, n_l$.

2. $\tau = t - 1, \ldots, t_0$. In this case,

$$\begin{aligned}\epsilon_{M,j}(\tau) &= \frac{\partial F}{\partial y_{M,j}(\tau)} \\ &= \frac{\partial F}{\partial y^*_{M,j}(\tau)} + \sum_{k=1}^{n_1} \frac{\partial F}{\partial s_{1,k}(\tau+1)} \frac{\partial s_{1,k}(\tau+1)}{\partial y_{M,j}(\tau)} + \sum_{k=1}^{n_M} \frac{\partial F}{\partial s_{M,k}(\tau+1)} \frac{\partial s_{M,k}(\tau+1)}{\partial y_{M,j}(\tau)} \\ &= -e_j(\tau) + \sum_{k=1}^{n_1} \delta_{1,k}(\tau+1) w_{M,j,1,k} + \sum_{k=1}^{n_M} \delta_{M,k}(\tau+1) w_{M,j,M,k}\end{aligned} \tag{22.14}$$

where M means the output layer M of network and $j \in \{1, 2, \ldots, n_M\}$,

$$\begin{aligned}
\epsilon_{l,j}(\tau) = \frac{\partial F}{\partial y_{l,j}(\tau)} &= \sum_{k=1}^{n_l} \frac{\partial F}{\partial s_{l,k}(\tau+1)} \frac{\partial s_{l,k}(\tau+1)}{\partial y_{l,j}(\tau)} + \sum_{m=1}^{n_{l+1}} \frac{\partial F}{\partial s_{l+1,m}(\tau)} \frac{\partial s_{l+1,m}(\tau)}{\partial y_{l,j}(\tau)} \\
&= \sum_{k=1}^{n_l} \delta_{l,k}(\tau+1) w_{l,j,l,k} + \sum_{m=1}^{n_{l+1}} \frac{\partial F}{\partial y_{l+1,m}(\tau)} \frac{\partial y_{l+1,m}(\tau)}{\partial s_{l+1,m}(\tau)} \frac{\partial s_{l+1,m}(\tau)}{\partial y_{l,j}(\tau)} \\
&= \sum_{m=1}^{n_l} \delta_{l,k}(\tau+1) w_{l,j,l,k} + \sum_{m=1}^{n_{l+1}} \epsilon_{l+1,m}(\tau) f'(s_{l+1,m}(\tau)) w_{l,j,l+1,m} \qquad (22.15)
\end{aligned}$$

where $l = 1, 2, \ldots, M-1$ and $j = 1, 2, \ldots, n_l$ and

$$\delta_{l,j}(\tau) = \frac{\partial F}{\partial s_{l,j}(\tau)} = \frac{\partial F}{\partial y_{l,j}(\tau)} \frac{\partial y_{l,j}(\tau)}{\partial s_{l,j}(\tau)} = \epsilon_{l,j}(\tau) f'(s_{l,j}(\tau)) \qquad (22.16)$$

where $l = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, n_l$.

In addition, for any appropriate i and j

$$\frac{\partial F}{\partial w_{i,j,l,k}} = \sum_{\tau=t_0}^{t} \frac{\partial F}{\partial w_{i,j,l,k}(\tau)} \qquad (22.17)$$

and for any τ

$$\frac{\partial F}{\partial w_{i,j,l,k}(\tau)} = \frac{\partial F}{\partial s_{l,k}(\tau)} \frac{\partial s_{l,k}(\tau)}{\partial w_{i,j,l,k}(\tau)} = \delta_{l,k}(\tau) y_{i,j}(\tau) \ or\ \delta_{l,k}(\tau) y_{i,j}(\tau-1) \qquad (22.18)$$

Combining these last two equations yields

$$\frac{\partial F}{\partial w_{i,j,l,k}} = \sum_{\tau=t_0}^{t} \delta_{l,k}(\tau) y_{i,j}(\tau) \ or\ \delta_{l,k}(\tau) y_{i,j}(\tau-1) \qquad (22.19)$$

Eqs. (22.11)–(22.16) and (22.19) represent the BPTT computation of $\partial F/\partial w_{i,j,l,k}$ for differentiable function F expressed in terms of the outputs of individual units in the network.

22.3.1.4 Real-Time Backpropagation through Time

In real-time BPTT, the performance measure is $J(t)$ at each time. To compute the gradient of $J(t)$ at time t, we proceed as follows. First, consider t fixed for the moment. This allows us the notational convenience of suppressing any reference to t in the following. Compute $\epsilon_{l,j}(\tau)$ and $\delta_{l,k}(\tau)$ for $\tau \in [t_0, t]$ by means of Eqs. (22.11)–(22.13). Eq. (22.14) needs a little change since with $F = J(t)$, $e_j(\tau) = 0$; thus, for $\tau < t$,

$$\begin{aligned}
\epsilon_{M,j}(\tau) = \frac{\partial F}{\partial y_{M,j}(\tau)} &= \frac{\partial F}{\partial y^*_{M,j}(\tau)} + \sum_{k=1}^{n_1} \frac{\partial F}{\partial s_{1,k}(\tau+1)} \frac{\partial s_{1,k}(\tau+1)}{\partial y_{M,j}(\tau)} + \sum_{k=1}^{n_1} \frac{\partial F}{\partial s_{M,k}(\tau+1)} \frac{\partial s_{M,k}(\tau+1)}{\partial y_{M,j}(\tau)} \\
&= \sum_{k=1}^{n_1} \frac{\partial F}{\partial s_{1,k}(\tau+1)} \frac{\partial s_{1,k}(\tau+1)}{\partial y_{M,j}(\tau)} + \sum_{k=1}^{n_1} \frac{\partial F}{\partial s_{M,k}(\tau+1)} \frac{\partial s_{M,k}(\tau+1)}{\partial y_{M,j}(\tau)} \\
&= \sum_{k=1}^{n_1} \delta_{1,k}(\tau+1) w_{M,j,1,k} + \sum_{k=1}^{n_M} \delta_{M,k}(\tau+1) w_{M,j,M,k} \qquad (22.20)
\end{aligned}$$

Thus, Eqs. (22.11)–(22.16), (22.19) and (22.20) represent the real-time BPTT. The process begins by using Eq. (22.11) to determine $\epsilon_{M,j}(t)$. This step is called *injecting error*, or to be more precise, *injecting* $e(t)$ at time t. Then δ and ϵ are obtained for successively earlier time steps through the repeated use of the Eqs. (22.15), (22.16), and (22.20). Here $\epsilon_{l,j}(\tau)$ represents the sensitivity of the *instantaneous performance* measure $J(t)$ to small perturbations in the output of the jth unit at layer l at time τ, while $\delta_{l,j}(\tau)$ represents the corresponding sensitivity to small perturbations to that unit's net input at that time. Once the backpropagation computation has been performed down to time t_0, the desired gradient of instantaneous performance is computed by the following pseudocode:

```
for $\tau = t$ to $t_0$ {
  for $l = 2$ to $M$ {
    for $k = 1$ to $n_l$ {
for $j = 0$ to $n_{l-1}$ {
      $\frac{\partial F}{\partial w_{l-1,j,l,k}} += \frac{\partial F}{\partial s_{l,k}(\tau)} \frac{\partial s_{l,k}(\tau)}{\partial w_{l-1,j,l,k}} = \delta_{l,k}(\tau) y_{l-1,j}(\tau)$    (22.21)
}
for $j = 1$ to $n_l$ {
      $\frac{\partial F}{\partial w_{l,j,l,k}} += \frac{\partial F}{\partial s_{l,k}(\tau)} \frac{\partial s_{l,k}(\tau)}{\partial w_{l,j,l,k}} = \delta_{l,k}(\tau) y_{l,j}(\tau - 1)$    (22.22)
}
} /*k loop*/ } /*l loop*/
for $k = 1$ to $n_1$ {
for $j = 1$ to $n^{ext}$ {
      $\frac{\partial F}{\partial w_{0,j,1,k}} += \frac{\partial F}{\partial s_{l,k}(\tau)} \frac{\partial s_{l,k}(\tau)}{\partial w_{0,j,1,k}} = \delta_{1,k}(\tau) y_j^{ext}(\tau)$    (22.23)
}
for $j = 1$ to $n_M$ {
      $\frac{\partial F}{\partial w_{M,j,1,k}} += \frac{\partial F}{\partial s_{l,k}(\tau)} \frac{\partial s_{l,k}(\tau)}{\partial w_{M,j,1,k}} = \delta_{1,k}(\tau) y_{M,j}(\tau - 1)$    (22.24)
}
for $j = 1$ to $n_1$ {
      $\frac{\partial F}{\partial w_{1,j,1,k}} += \frac{\partial F}{\partial s_{l,k}(\tau)} \frac{\partial s_{l,k}(\tau)}{\partial w_{1,j,1,k}} = \delta_{1,k}(\tau) y_{1,j}(\tau - 1)$    (22.25)
}
} /*k loop*/
} /*τ loop*/
```

where the notation "$+ =$" is to indicate that the quantity on the right-hand side of an expression is added to the previous value (time) on the left-hand side. Thus, the sum of $\frac{\partial F}{\partial w_{i,j,l,k}}$ from t_0 to t is computed. Because this algorithm makes use of potentially unbounded history storage, it is also sometimes called BPTT(∞).

22.3.2 Computational Capabilities of Discrete and Continuous Recurrent Networks

The computational power of recurrent nets is investigated in Refs. [15,16]; see also Ref. [63] for a thorough discussion of recurrent nets and analog computation in general. Recurrent nets include feedforward nets and thus the results of feedforward nets apply to recurrent nets as well. But recurrent nets gain considerably more computational power with increasing computation time. In the following, for the sake of concreteness, we assume that the piecewise-linear function

$$\pi(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } 0 \leq x \leq 1 \\ 1 & \text{if } x \geq 1 \end{cases}$$

is chosen as activation function. We concentrate on binary input and assume that the input is provided one bit at a time.

First of all, if weights and thresholds are integers, then each node computes a bit. Recurrent net with integer weights thus turn out to be equivalent to finite automata and they recognize exactly the class of regular language over the binary alphabet {0, 1}.

The computational power increases considerably for rational weights and thresholds. For instance, a "rational" recurrent net is, up to a polynomial-time computation, equivalent to a Turing machine. In particular, a network that simulates a universal Turing machine does exist and one could refer to such a network as "universal" in the Turing sense. It is important to note that the number of nodes in the simulating recurrent net is fixed (i.e., *does not grow* with increasing input length).

Irrational weights provide a further boost in computation power. If the net is allowed exponential computation time, then arbitrary Boolean functions (including noncomputable functions) are recognizable. However, if only polynomial computation time is allowed, then nets have less power and recognize exactly the languages computable by polynomial-size Boolean circuits.

References

[1] Hertz, J., Krogh, A., and Palmer, R. G., *Introduction to the Theory of Neural Computation*, Addison-Wesley, Reading, MA, 1991.

[2] Parberry, I., A primer on the complexity theory of neural networks, in *Formal Techniques in Artificial Intelligence: A Sourcebook,* Banerji, R. B., Ed., Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1990, p. 217.

[3] D. E. Rumelhart, J. L. McClelland and the PDP Reaserch Group, Parallel Distributed Processing - Vol. 1 and 2, The MIT Press, 1987.

[4] Siu, K.-Y., Roychowdhury, V., and Kailath, T., *Discrete Neural Computation: A Theoretical Foundation*, Prentice-Hall, Englewood Cliffs, NJ, 1994.

[5] Carpenter, G. A. and Grossberg, S., A massively parallel architecture for a self-organizing neural pattern recognition machine, *Comput. Vision, Graphics, Image Process.,* 37, 54, 1987.

[6] Giles, C. E., Sun, G. Z., Chen, H. H., Lee, Y. C., and Chen, D., Higher order recurrent networks and grammatical inference, in *Advances in Neural Information Processing Systems,* Vol. 2, Touretzky, D. S., Ed., Morgan Kaufmann, San Mateo, CA, 1990.

[7] Kawato, M., Furukawa, K., and Suzuki, R., A hierarchical neural-network model for control and learning of voluntary movement, *Biol. Cybern.*, 57, 169, 1987.

[8] Widrow, B., Winter, R. G., and Baxter, R. A., Layered neural nets for pattern recognition, *IEEE Trans. Acoust., Speech Signal Process.,* 36, 1109, 1988.

[9] Maass, W., Perspectives of current research about the complexity of learning in neural nets, in *Theoretical Advances in Neural Computation and Learning,* Roychowdhury, V. P., Siu, K. Y., and Orlitsky, A., Eds., Kluwer Academic Publishers, Dordrecht, 1994, p. 295.

[10] Arai, W., Mapping abilities of three-layer networks, *Proc. Int. Joint Conf. on Neural Networks,* 1989, p. 419.

[11] Cybenko, G., Approximation by superposition of a sigmoidal function, *Math. Control, Signals, Syst.,* 2, 303, 1989.

[12] Gallant, A. R. and White, H., There exists a neural network that does not make avoidable mistakes, *Proc. Int. Joint Conf. on Neural Networks,* 1988, p. 657.

[13] Mhaskar, H. N., Approximation by superposition of sigmoidal and radial basis functions, *Adv. Appl. Math.,* 13, 350, 1992.

[14] Poggio, T. and Girosi, F., A Theory of Networks for Approximation and Learning, Artificial Intelligence Memorandum, No. 1140, 1989.

[15] Siegelmann, H. and Sontag, E. D., Analog computation, neural networks, and circuit, *Theor. Comput. Sci.,* 131, 331, 1994.

[16] Siegelmann, H. and Sontag, E. D., On the computational power of neural nets, *JCSS,* 50, 132, 1995.

[17] Kolmogorov, A. N., On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition, *Dokl. Akad. Nauk USSR,* 114, 953, 1957.

[18] Barron, A. R., Approximation and estimation bounds for artificial neural networks, *Proc. of COLT,* Morgan Kaufmann, San Mateo, CA, 1991, p. 243.

[19] Darken, C., Donahue, M., Gurvits, L., and Sontag, E. D., Rate of approximation results motivated by robust neural network learning, *Proc. of COLT,* 1993, p. 303.

[20] DasGupta, B. and Schnitger, G., Analog versus discrete neural networks, *Neural Comput.,* 8(4), 805, 1996.

[21] Baum, E. B. and Haussler, D., What size net gives valid generalization? *Neural Comput.,* 1, 151, 1989.

[22] DasGupta, B. and Sontag, E. D., Sample complexity for learning recurrent perceptron mappings, *IEEE Trans. Inf. Theor.,* 42(5), 1479, 1996.

[23] Maass, W., Bounds for the computational power and learning complexity of analog neural nets, *Proc. of STOC,* 1993, p. 335.

[24] Goldberg, P. and Jerrum, M., Bounding the Vapnik–Chervonenkis dimension of concept classes parameterized by real numbers, *Proc. of COLT,* 1993, p. 361.

[25] Macintyre, A. and Sontag, E. D., Finiteness results for sigmoidal 'neural' networks, *Proc. of STOC,* 1993, p. 325.

[26] Judd, J. S., On the complexity of learning shallow neural networks, *J. Complexity,* 4, 177, 1988.

[27] Judd, J. S., *Neural Network Design and the Complexity of Learning,* MIT Press, Cambridge, MA, 1990.

[28] Blum, A. and Rivest, R. L., Training a 3-node neural network is NP-complete, *Neural Net.,* 5, 117, 1992.

[29] Lin, J.-H. and Vitter, J. S., Complexity results on learning by neural networks, *Machine Learn.,* 6, 211, 1991.

[30] Goldmann, M. and Hastad, J., On the power of small-depth threshold circuits, *Comput. Complexity,* 1(2), 113, 1991.

[31] Hajnal, A., Maass, W., Pudlak, P., Szegedy, M., and Turan, G., Threshold circuits of bounded depth, *Proc. of FOCS,* 1987, p. 99.

[32] Reif, J. H., On threshold circuits and polynomial computation, *Proc. 2nd Annual Structure in Complexity Theory,* 1987, p. 118.

[33] Hopfield, J. J., Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. National Academy of Science USA,* 1984, p. 3008.

[34] Minsky, M. and Papert, S., *Perceptrons,* The MIT Press, Cambridge, MA, Expanded edition, 1988.

[35] Littlestone, N., Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm, *Proc. of FOCS,* 1987, p. 68.

[36] Parberry, I. and Schnitger, G., Parallel computation with threshold functions, *JCSS,* 36(3), 278, 1988.

[37] Paturi, R. and Saks, M. E., On threshold circuits for parity, *Proc. of FOCS,* 1990, p. 397.

[38] Roychowdhury, V. P., Orlitsky, A., and Siu, K. Y., Lower bounds on threshold and related circuits via communication complexity, *IEEE Transactions on Information Theory,* Vol. 40, No. 2, pp. 467–474, March 1994.

[39] Dasgupta, B. and Schnitger, G., The power of approximating: a comparison of activation functions, in *Advances in Neural Information Processing Systems,* Vol. 5, Giles, C. L., Hanson, S. J., and Cowan, J. D., Eds., Morgan Kaufmann, San Mateo, CA, 1993, p. 615.

[40] Maass, W., Schnitger, G., and Sontag, E. D., On the computational power of sigmoid versus boolean threshold circuits, *Proc. of FOCS,* 1991, p. 767.

[41] Werbos, P. J., Backpropagation through time: what it does and how to do it, in *The Roots of Backpropagation: From Ordered Derivatives to Neural Network and Political Forecasting,* Wiley, New York, 1994, p. 269.

[42] Teasuro, G., He, Y., and Ahmad, S., Asymptotic convergence of backpropagation, *Neural Comput.,* 1, 382, 1989.

[43] Teasuro, G. and Janssens, B., Scaling relationships in back-propagation learning, *Complex Syst.*, 2, 39, 1988.

[44] Brady, M., Raghavan, R., and Slawny, J., Backpropagation fails to separate where perceptrons succeed, *IEEE Trans. Circuits Syst.*, 26, 665, 1989.

[45] Shrivastave, Y. and Dasgupta, S., Convergence issues in perceptron based adaptive neural network models, *Proc. 25th Allerton Conference on Communication, Control and Computing,* 1987, p. 1133.

[46] Wittner, B. S. and Denker, J. S., Strategies for teaching layered networks classification tasks, in *Proc. of Conference on Neural Information Processing Systems,* Anderson, D., Ed., American Institute of Physics, New York, 1987.

[47] Vapnik, V. N., *Estimation of Dependencies Based on Empirical Data*, Springer, Berlin, 1982.

[48] Vidyasagar, M., *Learning and Generalization with Applications to Neural Networks*, Springer, London, 1996.

[49] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M., Learnability and the Vapnik–Chervonenkis dimension, *JACM,* 36, 929, 1989.

[50] Vapnik, V. N. and Chervonenkis, A. Ja., *Theory of Pattern Recognition (in Russian)*, Moscow, Nauka, 1974. (German translation: Wapnik, W. N. and Chervonenkis, A. Ja., *Theorie der Zeichenerkennung*, Akademia-Verlag, Berlin, 1979)

[51] Valiant, L. G., A theory of the learnable, *CACM,* 27, 1134, 1984.

[52] Cover, T. M., Geometrical and statistical properties of linear threshold elements, Ph.D. thesis, Stanford, 1964, Stanford SEL TR 6107-1, May 1964.

[53] Cover, T. M., Capacity problems for linear machines, in *Pattern Recognition,* Kanal, L., Ed., Thompson Book Company, Washington, DC, 1968, p. 283.

[54] Karpinski, M. and Macintyre, A., Polynomial bounds for VC dimension of sigmoidal and general Pfaffian neural networks, *JCSS,* 54, 169, 1997.

[55] Koiran, P. and Sontag, E. D., Neural networks with quadratic VC dimension, *JCSS,* 54, 190, 1997.

[56] DasGupta, B., Siegelmann, H. T., and Sontag, E. D., On a learnability question associated to neural networks with continuous activations, *Proc. of COLT,* 1994, p. 47.

[57] DasGupta, B., Siegelmann, H. T., and Sontag, E. D., On the complexity of training neural networks with continuous activation functions, *IEEE Trans. Neural Netw.,* 6(6), 1490, 1995.

[58] Höffgen, K.-U., Computational limitations on training sigmoidal neural networks, *Inf. Process. Lett.*, 46, 269, 1993.

[59] Kearns, M., Li, M., Pitt, L., and Valiant, L., On the learnability of boolean formulae, *Proc. of STOC,* 1987, p. 285.

[60] Blum, A. L. and Kannan, R., Learning and intersection of k halfspaces over a uniform distribution, *Proc. of FOCS,* 1993, p. 312.

[61] Prokhorov, D. V., Backpropagation through time and derivative adaptive critics—a common framework for comparison, in *Handbook of Learning and Approximate Dynamic Programming*, Si, J., Barto, A. G., Powell, W. B., and Wunsch, D., Eds., IEEE Press, 2004, p. 376.

[62] Willams, R. J. and Zipser, D., Gradient-based learning algorithm for recurrent networks and their computational complexity, in *Backpropagation: Theory, Architectures, and Applications*, Chauvin, Y. and Rumelhart, D. E., Eds., Lawrence Erlbaum Associates, Hillsdale, NJ, 1994, p. 433.

[63] Siegelmann, H. T., *Neural Networks and Analog Computation: Beyond the Turing Limit*, Birkhäuser, Boston, MA, 1998.

[64] Rumelhart, D. E., McClelland, J. L., and the PDP Research Group, Parallel Distributed Processing—Vol. 1 and 2, The MIT Press, 1987.

23

Principles of Tabu Search

Fred Glover
University of Colorado

Manuel Laguna
University of Colorado

Rafael Martí
University of Valencia

23.1 Introduction

Tabu search (TS) is a metaheuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality. The term *tabu search* was coined in the same paper that introduced the term *metaheuristic* [1]. Tabu search is based on the premise that problem solving, to qualify as intelligent, must incorporate *adaptive memory* and *responsive exploration*. The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memoryless designs that heavily rely on semirandom processes that implement a form of sampling. The emphasis on responsive exploration (and hence purpose) in TS, whether in a deterministic or probabilistic implementation, derives from the supposition that a bad strategic choice can often yield more information than a good random choice. Over a wide range of problem settings, strategic use of memory can make dramatic differences in the ability to solve problems.

Tabu search can be directly applied to virtually any kind of optimization problem. We can state most of these problems in the following form, where "optimize" means to minimize or maximize:

$$\begin{array}{ll} \text{Optimize} & f(x) \\ \text{subject to} & x \in \mathrm{X} \end{array}$$

The function $f(x)$ may be linear, nonlinear, or even stochastic, and the set $\mathbf{X}$ summarizes constraints on the vector of decision variables x. The constraints may similarly include linear, nonlinear, or stochastic inequalities, and may compel all or some components of x to receive discrete values.

While this representation is useful for discussing a number of problem-solving considerations, we emphasize that in many applications of combinatorial optimization, the problem of interest may not be easily formulated as an objective function subject to a set of constraints. The requirement $x \in \mathbf{X}$, for example, may specify logical conditions or interconnections that would be cumbersome to formulate mathematically, but may be better left as verbal stipulations that can be then coded as rules.

The TS technique is rapidly becoming the method of choice for designing solution procedures for hard combinatorial optimization problems. A comprehensive examination of this methodology can be found in the book by Glover and Laguna [2]. Widespread successes in practical applications of optimization have spurred a rapid growth of the method as a means of identifying extremely high-quality solutions

TABLE 23.1 Illustrative Tabu Search Applications

Scheduling	**Telecommunications**
Flow-time cell manufacturing	Call routing
Heterogeneous processor scheduling	Bandwidth packing
Workforce planning	Hub facility location
Rostering	Path assignment
Machine scheduling	Network design for services
Flow shop scheduling	Customer discount planning
Job shop scheduling	Failure immune architecture
Sequencing and batching	Synchronous optical networks
Design	**Production, Inventory, and Investment**
Computer-aided design	Supply chain management
Fault tolerant networks	Flexible manufacturing
Transport network design	Just-in-time production
Architectural space planning	Capacitated MRP
Diagram coherency	Part selection
Fixed charge network design	Multiitem inventory planning
Irregular cutting problems	Volume discount acquisition
Layout planning	Project portfolio optimization
Logic and Artificial Intelligence	**Routing**
Maximum satisfiability	Vehicle routing
Probabilistic logic	Capacitated routing
Pattern recognition/classification	Time window routing
Data mining	Multimode routing
Clustering	Mixed fleet routing
Statistical discrimination	Traveling salesman
Neural network training	Traveling purchaser
Neural network design	Convoy scheduling
Location and Allocation	**Graph Optimization**
Multicommodity location/allocation	Graph partitioning
Quadratic assignment	Graph coloring
Quadratic semiassignment	Clique partitioning
Multilevel generalized assignment	Maximum clique problems
Large-scale GAP problems	Maximum planner graphs
Technology	**General Combinational Optimization**
Seismic inversion	Zero–one programming
Electrical power distribution	Fixed charge optimization
Engineering structural design	Nonconvex nonlinear programming
Minimum volume ellipsoids	All-or-none networks
Space station construction	Bilevel programming
Circuit cell placement	Multiobjective discrete optimization
OffShore oil exploration	General mixed integer optimization

efficiently. Tabu search methods have also been used to create hybrid procedures with other heuristic and algorithmic methods, to provide improved solutions to problems in production planning and scheduling, resource allocation, network design, routing, financial analysis, telecommunications, portfolio planning, supply chain management, agent-based modeling, business process design, forecasting, machine learning, data mining, biocomputation, molecular design, forest management and resource planning, and many other areas. Some of the diversity of TS applications is shown in Table 23.1.

The TS emphasis on adaptive memory makes it possible to exploit the types of strategies that underlie the best of human problem-solving, instead of being confined to mimicking the processes found in lower orders of natural phenomena and behavior. The basic elements of TS have several important features, summarized in Table 23.2. Tabu search is concerned with finding new and more effective ways of taking advantage of the concepts embodied in Table 23.2, and with identifying associated principles that can expand the foundations of intelligent search.

TABLE 23.2 Principal Tabu Search Features

Adaptive Memory
Selectivity (including strategic forgetting)
Abstraction and decomposition (through explicit and attributive memory)
Timing Recency of events Frequency of events Differentiation between short term and long term
Quality and impact Relative attractiveness of alternative choices Magnitude of changes in structure or constraining Relationships
Context Regional interdependence Structural interdependence Sequential interdependence
Responsive Exploration
Strategically imposed restraints and inducements (*tabu conditions* and *aspiration levels*)
Concentrated focus on good regions and good solution features (*intensification processes*)
Characterizing and exploring promising new regions (*diversification processes*)
Nonmonotonic search patterns (*strategic oscillation*)
Integrating and extending solutions (*path relinking*)

In this chapter we will describe some key aspects of this methodology, as the use of memory structures and search strategies, and illustrate them in an implementation to solve the linear ordering problem (LOP).

23.2 Memory Structures

Tabu search begins in the same way as ordinary local or neighborhood search, proceeding iteratively from one point (solution) to another until a chosen termination criterion is satisfied. Each solution x has an associated neighborhood $N(x) \subset X$, and each solution $x' \in N(x)$ is reached from x by an operation called a *move*.

We may contrast TS with a simple descent method where the goal is to minimize $f(x)$. Such a method only permits moves to neighbor solutions that improve the current objective function value and ends when no improving solutions can be found. The final x obtained by a descent method is called a local optimum, since it is at least as good as or better than all solutions in its neighborhood. The evident shortcoming of a descent method is that such a local optimum in most cases will not be a global optimum, that is, it usually will not minimize $f(x)$ over all $x \in \mathbf{X}$.

Tabu search permits moves that deteriorate the current objective function value but the moves are chosen from a modified neighborhood $N^*(x)$. Short- and long-term memory structures are responsible for the specific composition of $N^*(x)$. In other words, the modified neighborhood is the result of maintaining a selective history of the states encountered during the search. In the TS strategies based on short-term considerations, $N^*(x)$ characteristically is a subset of $N(x)$, and the tabu classification serves to identify elements of $N(x)$ excluded from $N^*(x)$. In TS strategies that include longer term considerations, $N^*(x)$ may also be expanded to include solutions not ordinarily found in $N(x)$, such as

TABLE 23.3 Examples of Recency-Based Memory

Context	Attributes	To Record the Last Time ...
Binary problems	Variable index (i)	Variable i changed its value from 0 to 1 or 1 to 0 (depending on its current value)
Job sequencing	Job index (j) Job index (j) and position (p) Pair of job indexes (i, j)	Job j changed positions Job j occupied position p Job i exchange positions with job j
Graphs	Arc index (i)	Arc i was added to the current solution Arc i was dropped from the current solution

solutions found and evaluated in past search, or identified as high-quality neighbors of these past solutions. Characterized in this way, TS may be viewed as a dynamic neighborhood method. This means that the neighborhood of x is not a static set, but rather a set that can change according to the history of the search.

The structure of a neighborhood in TS differs from that used in local search in an additional manner, by embracing the types of moves used in constructive and destructive processes (where the foundations for such moves are accordingly called *constructive neighborhoods* and *destructive neighborhoods*). Such expanded uses of the neighborhood concept reinforce a fundamental perspective of TS, which is to define neighborhoods in dynamic ways that can include serial or simultaneous consideration of multiple types of moves.

Tabu search uses attributive memory for guiding purposes (i.e., to compute $N^*(x)$). Instead of recording full solutions, attributive memory structures are based on recording attributes. This type of memory records information about solution properties (attributes) that change in moving from one solution to another. The most common attributive memory approaches are recency- and frequency-based memory. Recency, as its name suggests, keeps track of solutions attributes that have changed during the recent past. Frequency typically consists of ratios about the number of iterations a certain attribute has changed or not (depending whether it is a transition or a residence frequency). Some examples of recency- and frequency-based memory are shown in Table 23.3 and Table 23.4 respectively.

Characteristically, a TS process based strictly on short-term strategies may allow a solution x to be visited more than once, but it is likely that the corresponding reduced neighborhood $N^*(x)$ will be different each time. With the inclusion of longer term considerations, the likelihood of duplicating a previous neighborhood upon revisiting a solution, and more generally of making choices that repeatedly visit only a limited subset of X, is all but nonexistent.

Recency-based memory is the most common memory structure used in TS implementations. As its name suggests, this memory structure keeps track of solutions attributes that have changed during the recent past. To exploit this memory, selected attributes that occur in solutions recently visited are labeled

TABLE 23.4 Examples of Frequency-Based Memory

Context	Residence Measure	Transition Measure
Binary problems	Number of times variable i has been assigned the value of 1	Number of times variable i has changed values
Job sequencing	Number of times job j has occupied position p Average objective function value when job j occupies position p	Number of times job i has exchanged positions with job j Number of times job j has been moved to an earlier position in the sequence
Graphs	Number of times arc i has been part of the current solution Average objective function value when arc i is part of the solution	Number of times arc i has been deleted from the current solution when arc j has been added Number of times arc i has been added during improving moves

tabu-active, and solutions that contain tabu-active elements, or particular combinations of these attributes, are those that become tabu. This prevents certain solutions from the recent past from belonging to $N^*(x)$ and hence from being revisited. Other solutions that share such tabu-active attributes are also similarly prevented from being visited. Note that while the tabu classification strictly refers to solutions that are forbidden to be visited, by virtue of containing tabu-active attributes (or more generally by violating certain restriction based on these attributes), moves that lead to such solutions are also often referred to as being tabu.

Frequency-based memory provides a type of information that complements the information provided by recency-based memory, broadening the foundation for selecting preferred moves. Like recency, frequency often is weighted or decomposed into subclasses. Also, frequency can be integrated with recency to provide a composite structure for creating penalties and inducements that modify move evaluations.

Frequencies typically consist of ratios, whose numerators represent counts expressed in two different measures: a *transition measure*—the number of iterations where an attribute changes (enters or leaves) the solutions visited on a particular trajectory, and a *residence measure*—the number of iterations where an attribute belongs to solutions visited on a particular trajectory, or the number of instances where an attribute belongs to solutions from a particular subset. The denominators generally represent one of three types of quantities: (1) the total number of occurrences of all events represented by the numerators (such as the total number of associated iterations); (2) the sum (or average) of the numerators; and (3) the maximum numerator value. In cases where the numerators represent weighted counts, some of which may be negative, denominator (3) is expressed as an absolute value and denominator (2) is expressed as a sum of absolute values (possibly shifted by a small constant to avoid a zero denominator). The ratios produce *transition frequencies* that keep track of how often attributes change, and *residence frequencies* that keep track of how often attributes are members of solutions generated. In addition to referring to such frequencies, thresholds based on the numerators alone can be useful for indicating when phases of greater diversification are appropriate.

23.3 Search Strategies

The use of recency and frequency memory in TS generally fulfills the function of preventing searching processes from *cycling*, that is, from endlessly executing the same sequence of moves (or more generally, from endlessly and exclusively revisiting the same set of solutions). More broadly, however, the various manifestations of these types of memory are designed to impart additional robustness or vigor to the search.

A key element of the adaptive memory framework of TS is to create a balance between search intensification and diversification. Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Diversification strategies, however, seek to incorporate new attributes and attribute combinations that were not included within solutions generated previously. In one form, these strategies undertake to drive the search into regions dissimilar to those already examined. It is important to keep in mind that intensification and diversification are not mutually opposing, but are rather mutually reinforcing.

Most types of intensification strategies require a means for identifying a set of elite solutions as basis for incorporating good attributes into newly created solutions. Membership in the elite set is often determined by setting a threshold that is connected to the objective function value of the best solution found during the search. A simple instance of the intensification strategy is shown in Figure 23.1. Two simple variants for elite solution selection have proved quite successful. One introduces a diversification measure to assure the solutions recorded differ from each other by a desired degree, and then erases all short-term memory before resuming from the best of the recorded solutions. The other keeps a bounded length sequential list that adds a new solution at the end only if it is better than any previously seen, and the short-term memory that accompanied this solution is also saved.

```
Apply TS short-term memory
Apply an elite selection strategy.
do {
        Choose one of the elite solutions.
        Resume short-term memory TS from chosen solution.
        Add new solutions to elite list when applicable.
} while (iterations < limit and list not empty)
```

FIGURE 23.1 Simple TS intensification approach.

Diversification is automatically created in TS (to some extent) by short-term memory functions, but is particularly reinforced by certain forms of longer term memory. TS diversification strategies are often based on modifying choice rules to bring attributes into the solution that are infrequently used. Alternatively, they may introduce such attributes by periodically applying methods that assemble subsets of these attributes into candidate solutions for continuing the search, or by partially or fully restarting the solution process. Diversification strategies are particularly helpful when better solutions can be reached only by crossing barriers or "humps" in the solution space topology.

The incorporation of modified choice rules can be moderated by using the following penalty function:

$$MoveValue' = MoveValue + d * Penalty$$

This type of penalty approach is commonly used in TS, where the *Penalty* value is often a function of frequency measures such as those indicated in Table 23.2, and *d* is an adjustable diversification parameter. Larger *d* values correspond to a desire for more diversification.

23.4 Advanced Designs: Strategic Oscillation and Path Relinking

There are many forms in which a simple TS implementation can be improved by adding long-term elements. In this paper we restrict our attention to two of the most used methods, namely strategic oscillation and path relinking (PR), which constitute the core of many adaptive memory programming algorithms.

Strategic oscillation operates by orienting moves in relation to a critical level, as identified by a stage of construction or a chosen interval of functional values. Such a critical level or *oscillation boundary* often represents a point where the method would normally stop. Instead of stopping when this boundary is reached, however, the rules for selecting moves are modified, to permit the region defined by the critical level to be crossed. The approach then proceeds for a specified depth beyond the oscillation boundary, and turns around. The oscillation boundary again is approached and crossed, this time from the opposite direction, and the method proceeds to a new turning point (see Figure 23.2).

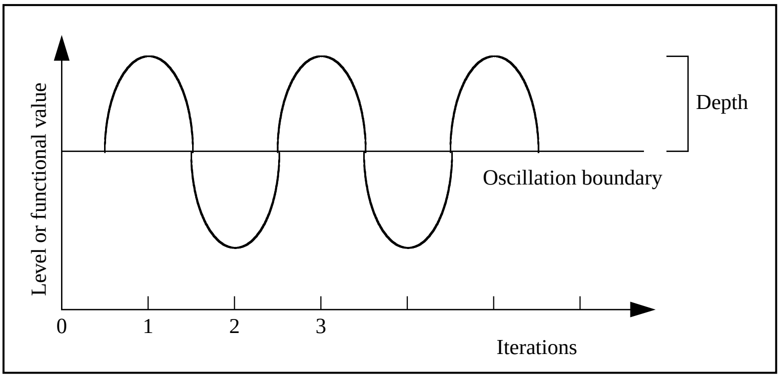


FIGURE 23.2 Strategic oscillation.

The process of repeatedly approaching and crossing the critical level from different directions creates an oscillatory behavior, which gives the method its name. Control over this behavior is established by generating modified evaluations and rules of movement, depending on the region navigated and the direction of search. The possibility of retracing a prior trajectory is avoided by standard TS mechanisms, like those established by the recency-based and frequency-based memory functions.

When the level or functional values in Figure 23.2 refer to degrees of feasibility and infeasibility, a vector-valued function associated with a set of problem constraints can be used to control the oscillation. In this case, controlling the search by bounding this function can be viewed as manipulating a parameterization of the selected constraint set. A preferred alternative is often to make the function a Lagrangean or surrogate constraint penalty function, avoiding vector-valued functions and allowing tradeoffs between degrees of violation of different component constraints.

Path relinking, as a strategy of creating trajectories of moves passing through high-quality solutions was first proposed in connection with TS by Glover [3]. The approach was then elaborated in greater detail as a means of integrating TS intensification and diversification strategies, and given the name PR by Glover and Laguna [4]. Path relinking generally operates by starting from an *initiating solution*, selected from a subset of high-quality solutions, and generating a path in the neighbourhood space that leads toward the other solutions in the subset, which are called *guiding solutions.* This is accomplished by selecting moves that introduce attributes contained in the guiding solutions.

Path relinking can be considered an extension of the combination method of scatter search [4,5]. Instead of directly producing a new solution when combining two or more original solutions, PR generates paths between and beyond the selected solutions in the neighborhood space. The character of such paths is easily specified by reference to solution attributes that are added, dropped, or otherwise modified by the moves executed. Examples of such attributes include edges and nodes of a graph, sequence positions in a schedule, vectors contained in linear programming basic solutions, and values of variables and functions of variables.

The approach may be viewed as an extreme (highly focused) instance of a strategy that seeks to incorporate attributes of high-quality solutions, by creating inducements to favor these attributes in the moves selected. However, instead of using an inducement that merely encourages the inclusion of such attributes, the PR approach subordinates other considerations to the goal of choosing moves that introduce the attributes of the guiding solutions, to create a "good attribute composition" in the current solution. The composition at each step is determined by choosing the best move, using customary choice criteria, from a restricted set—the set of those moves currently available that incorporate a maximum number (or a maximum-weighted value) of the attributes of the guiding solutions. (Exceptions are provided by aspiration criteria, as subsequently noted.) The approach is called PR either by virtue of generating a new path between solutions previously linked by a series of moves executed during a search, or by generating a path between solutions previously linked to other solutions but not to each other.

To generate the desired paths, it is only necessary to select moves that perform the following role: upon starting from an *initiating solution,* the moves must progressively introduce attributes contributed by a *guiding solution* (or reduce the distance between attributes of the initiating and guiding solutions). The roles of the initiating and guiding solutions are interchangeable; each solution can also be induced to move simultaneously toward the other as a way of generating combinations. First, consider the creation of paths that join two selected solutions x' and x'', restricting attention to the part of the path that lies "between" the solutions, producing a solution sequence $x' = x(1), x(2), \ldots, x(r) = x''$. To reduce the number of options to be considered, the solution $x(i+1)$ may be created from $x(i)$ at each step by choosing a move that minimizes the number of moves remaining to reach x''. The relinked path may encounter solutions that may not be better than the initiating or guiding solution, but that provide fertile "points of access" for reaching other, somewhat better, solutions. For this reason it is valuable to examine neighboring solutions along a relinked path, and keep track of those of high quality that may provide a starting point for launching additional searches.

As described by Martí et al. [6], we can apply different PR elements to perform more elaborated designs. Some examples are simultaneous relinking, tunneling strategy, extrapolated relinking, multiple guiding solutions, constructive neighborhoods, or vocabulary building.

23.5 The Linear-Ordering Problem

Given a matrix of weights $E = \{e_{ij}\}_{m\times m}$, the LOP consists of finding a permutation p of the columns (and rows) to maximize the sum of the weights in the upper triangle. In mathematical terms, we seek to maximize

$$C_E(p) = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} e_{p_i p_j}$$

where p_i is the index of the column (and row) in position i in the permutation. Note that in the LOP, the permutation p provides the ordering of both the columns and the rows. Solution methods for this NP-hard problem have been proposed since 1958, when Chenery and Watanabe outlined some ideas on how to obtain solutions for this problem [7]. In this section we describe a TS implementation by Laguna et al. [8] for the LOP.

The LOP has a wide range of applications in several fields. Perhaps, the best known application occurs in the field of economics. In this application, the economy (regional or national) is first subdivided into sectors. Then, an input/output matrix is created, in which the entry (i, j) represents the flow of money from sector i to sector j. Economists are often interested in ordering the sectors so that suppliers tend to come first followed by consumers. This is achieved by permuting the rows and columns of the matrix so that the sum of entries above the diagonal is maximized, which is the objective of the LOP.

Insertions are used as the primary mechanism to move from one solution to another in Laguna et al.'s method for the LOP. *INSERT_MOVE*(p_j, i) consist of deleting p_j from its current position j to be inserted in position i (i.e., between the current sectors p_{i-1} and p_i). This operation results in the ordering p', as follows:

$$p' = \begin{cases} (p_1, \ldots, p_{i-1}, p_j, p_i, \ldots, p_{j-1}, p_{j+1}, \ldots, p_m) & \text{for } i < j \\ (p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_{i-1}, p_j, p_i, \ldots, p_m) & \text{for } i > j \end{cases}$$

The neighborhood N consists of all permutations resulting from executing general insertion moves as

$$N = \{p' : \mathit{INSERT\_MOVE}(p_j, i), \mathit{for}\, j = 1, \ldots, m\, \mathit{and}\, i = 1, 2, \ldots, j-1, j+1, \ldots, m\},$$

and N is partitioned into m neighborhoods, N^j, associated with each sector p_j, for $j = 1, \ldots, m$:

$$N^j = \{p' : \mathit{INSERT\_MOVE}(p_j, i), i = 1, 2, \ldots, j-1, j+1, \ldots, m\}$$

Starting from a randomly generated permutation p, the basic TS procedure alternates between an intensification and a diversification phase. An iteration of the *intensification phase* begins by randomly selecting a sector. The probability of selecting sector j is proportional to its weight w_j according to

$$w_j = \sum_{i\neq j} (e_{ij} + e_{ji})$$

The move INSERT_MOVE$(p_j, i) \in N^j$ with the largest move value is selected. (Note that this rule may result in the selection of a nonimproving move.) The move is executed even when the move value is not positive, resulting in a deterioration of the current objective function value. The moved sector becomes tabu-active for *TabuTenure* iterations, and therefore it cannot be selected for insertions during this time.

The number of times that sector j has been chosen to be moved is accumulated in the value *freq*(j). This frequency information is used for diversification purposes. The intensification phase terminates after *MaxInt* consecutive iterations without improvement. Before abandoning this phase, a local search procedure based in the same neighborhood is applied to the best solution found (during the current intensification). We denote this solution as $p^{\#}$, in contrast to p^* (the best solution found over the entire search). By applying this greedy procedure (without tabu restrictions), a local optimum is guaranteed as the output of the intensification phase.

The *diversification phase* is performed for *MaxDiv* iterations. At each iteration, a sector is randomly selected, where the probability of selecting sector j is inversely proportional to the frequency count *freq*(j). The chosen sector is placed in the best position, as determined by the move values associated with the insert moves in N^j. The procedure stops when *MaxGlo* global iterations are performed without improving $C_E(p^*)$. A global iteration is an application of the intensification phase followed by the application of the diversification phase.

An additional intensification is introduced by implementing a long-term PR phase. Specifically, the best solution found at the end of an intensification phase $p^{\#}$ (which does not necessarily represent p^*, the best solution overall) is subjected to a relinking process. The process consists of making moves starting from $p^{\#}$ (the initiating solution) in the direction of a set of elite solutions (also referred to as guiding solutions). The set of elite solutions consists of the *EltSol* best solutions found during the entire search. The insertions used to move the initiating solution closer to the guiding solutions can be described as follows. For each sector p_j in the current solution:

(1) Find the position i for which the absolute value of $(j\text{-}i)$ is minimized, where i is the position that p_j occupies in at least one of the guiding solutions.
(2) Perform *INSERT_MOVE*(p_j, i).

A long-term diversification phase is also implemented to complement the diversification phase in the basic procedure. The long-term diversification is applied after *MaxLong* global iterations have elapsed without improving $C_E(p^*)$. For each sector p_j, a rounded average position $\alpha(p_j)$ is calculated using the positions occupied by this sector in the set of elite solutions and the solutions visited during the last intensification phase. Then, m diversification steps are performed which insert each sector p_j in its complementary position $m\text{-}\alpha(p_j)$, that is, *INSERT_MOVE*(p_j, $m\text{-}\alpha(p_j)$) is executed for $j = 1, \ldots, m$.

After preliminary experimentation, the search parameters are set to *MaxGlo* $= 100$, *MaxLong* $= 50$, *EltSol* $= 4$, *TabuTenure* $= 2\sqrt{m}$, *MaxInt* $= m$, and *MaxDiv* $= 0.5m$ and *EltSol* $= 4$. In the 49 instances of the public domain library of linear ordering problems (LOLIB [15]), the method obtains the optimal solution within 1 s of computer time run on a Pentium IV at 3 GHz. The method is also compared with a previous procedure due to Chanas and Kobylanski [9] and a greedy procedure based on the N local search. The methods were run in a way that the best solution found was reported every 0.5 s. These data points were used to generate the performance graph in Figure 23.3. The superior performance of TS_LOP is made evident by Figure 23.3.

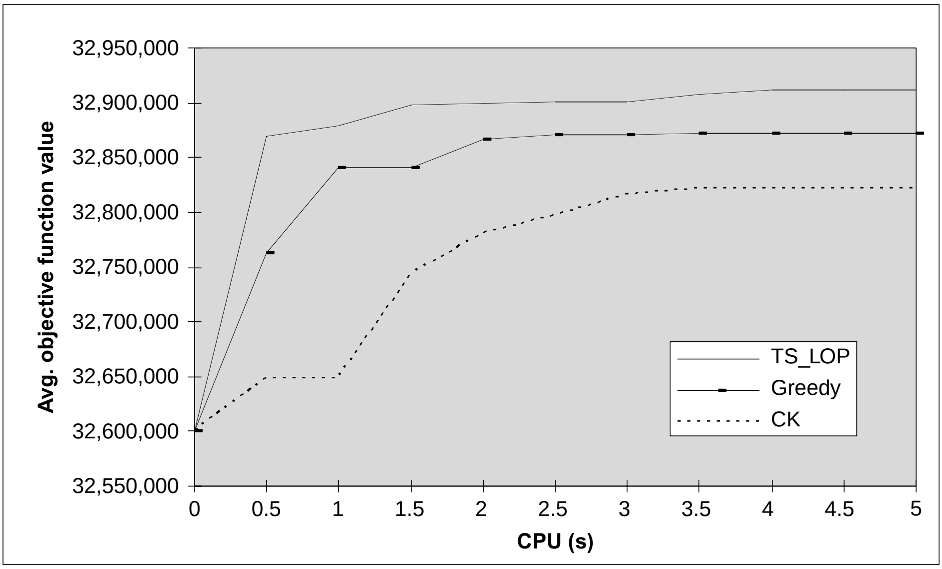


FIGURE 23.3 Performance graph.

23.6 The Tabu Cycle and Conditional Probability Methods

In this section, we describe the implementation and testing of the tabu cycle method and two variants of the conditional probability method [10]. These methods were originally described by Glover [11] and again in a book by Glover and Laguna [2], but have been largely ignored in the TS literature. The *tabu cycle method* is a short-term memory mechanism that is based on partitioning the elements (i.e., move attributes) of a tabu list. The methodology is general and capable of accommodating multiattribute TS memory, as described by Glover and Laguna [2]. In its most basic form, the tabu cycle method divides the short-term memory list into *TabuGroups* groups, where group k consists of elements that were added to the list between a specified range of iterations ago. While in some variants of TS (e.g., probabilistic TS) it is common to progressively relax the tabu status of elements as they become older, the tabu cycle method, by contrast, allows the elements of some groups to fully escape their tabu status according to certain frequencies that increase with the age of the groups. The method is based on the use of iteration intervals called *tabu cycles*, which are made smaller for older groups than for younger groups (with the exception of a small buffer group). Specifically, if group k has a tabu cycle of $TC(k)$ iterations, then at each occurrence of this number of iterations, on average, the elements of group k escape their tabu status and are free to be chosen. In other words, on average, group k is designated as FREE every $TC(k)$ iterations. Mechanisms and data structures that are useful for achieving this are described in Ref. [10].

The *conditional probability method* is a variant of the tabu cycle method that chooses elements by establishing the probability that a group will be FREE on a given iteration. The probability assigned to group k may be viewed conceptually as the inverse of the tabu cycle value. That is, $P(k) = 1/TC(k)$. Analogous to the tabu cycle method, group k is FREE if all older groups likewise are FREE. The method employs a *conditional probability*, $CP(k)$, as a means of determining whether a particular group k can be designated as FREE. The conditional probability values are fixed and the status of a group is determined by a probabilistic process that is not affected by previous choices. Consequently, the approach ignores the possibility that actual tabu cycle values may be far from their targets for some groups. This may happen, for example, when for a number of iterations no elements are chosen from a particular set of FREE groups. The conditional probability method also makes use of a buffer group, for which no element is allowed to escape its tabu status.

A variant of the conditional probability method uses substitute probability values to keep the expected number of elements per iteration chosen from groups no older than any given group k close to $P(k)$. The substitute probabilities replace the original $P(k)$ values in the determination of the conditional probabilities. These substitute probabilities make use of cycle counts, which are also used in connection with the tabu cycle method.

Laguna [10] uses a single machine scheduling problem to test the merit of implementations of the tabu cycle method and both variants of the conditional probability method. The problem consists of minimizing the sum of the setup costs and linear delay penalties when n jobs, arriving at time zero, are to be scheduled for sequential processing on a continuously available machine. Several variants of TS for this problem have been reported in the literature [12–14]. Experiments with more than 300 problem instances with up to 200 jobs were performed to compare a simple static and dynamic short-term memory schemes with a tabu cycle implementation (Cycle), a conditional probability implementation (C-Prob) and an implementation of the conditional probability method with substitute probabilities (S-Prob). The static short-term memory assigns a constant to the tabu tenure to all attributes during the search. The dynamic short-term memory randomly selects from a specified range a tabu tenure. Therefore, the tabu tenure assigned to an attribute in a given iteration may not be the same as the tabu tenure assigned to another attribute in a different iteration. Table 23.5 shows the number of best solutions found by each method in each set of 100 problems.

The results in Table 23.5 show the merit of the tabu cycle and the conditional probability variants as the problem size increases. In addition to these results, the S-Prob is able to find 17 new best solutions to 20 problems used for experimentation by Glover and Laguna [13]. For problems with up to 60 jobs,

TABLE 23.5 Number of Best Solutions (Out of 100) Found by Each Method

Problem Set	Static	Dynamic	Cycle	C-Prob	S-Prob
$n = 50$	2	50	9	31	65
$n = 100$	0	10	28	17	47
$n = 200$	0	8	37	26	29

for which a lower bound can be computed, S-Prob produces a maximum gap of 3.56% in relation to this optimistic bound.

These results confirm that a TS procedure based solely on a static tabu list is not a robust method, because it is incapable of maintaining an acceptable level of diversity during the search. The dynamic short-term memory continues to be an appealing alternative, because it is easy to implement and provides a good balance between diversification and intensification. The results also show that improved outcomes are possible with the additional effort required to implement the tabu cycle or conditional probability methods.

Additional strategies identified by Glover and Laguna [2] can be valuable for exploiting other aspects of intensification and diversification, but this example demonstrates the importance of handling short-term memory in a strategic way, especially when faced with larger and more difficult problems.

23.7 Conclusions

The focus and emphasis of TS have a number of implications for the goal of designing improved optimization procedures. These research opportunities carry with them an emphasis on producing systematic and strategically designed rules, rather than following the policy of relegating decisions to random choices, as often is fashionable in evolutionary methods. The highly attractive results provided by the adaptive memory structures underlying TS are producing an evident impact on the design of metaheuristic methods in general, and are motivating the emergence of new hybrids of TS with other procedures.

Acknowledgments

This research is partially supported by the Spanish Government under code TIN2006-02696.

References

[1] Glover, F., Future paths for integer programming and links to artificial intelligence, *Comput. Oper. Res.,* 13, 533, 1986.

[2] Glover, F. and Laguna, M., *Tabu Search,* Kluwer Academic Publishers, Dordrecht, 1997.

[3] Glover, F., Tabu search, Part I, *ORSA J. Comput.*, 1, 190, 1989.

[4] Glover, F. and Laguna, M., Tabu search, in *Modern Heuristic Techniques for Combinatorial Problems,* C. Reeves, Ed., Blackwell Scientific Publishing, Oxford, 1993, p. 70.

[5] Laguna, M. and Martí, R., *Scatter Search—Methodology and Implementations in C,* Kluwer Academic Publishers, Dordrecht, 2003.

[6] Martí, R., Laguna, M., and Glover, F., Principles of scatter search, *Eur. J. Oper. Res.*, 169, 359–372, 2004.

[7] Reinelt, G., The linear ordering problem: algorithms and applications, in *Research and Exposition in Mathematics*, Vol. 8, Hofmann, H. H., and Wille, R., Eds., Heldermann Verlag, Berlin, 1985.

[8] Laguna, M., Martí, R., and Campos, V., Intensification and diversification with elite tabu search solutions for the linear ordering problem, *Comput. Oper. Res.*, 26, 1217, 1999.

[9] Chanas, S. and Kobylanski, P., A new heuristic algorithm solving the linear ordering problem, *Comput. Optimization Appl.*, 6, 191, 1996.

[10] Laguna, M., Implementing and testing the tabu cycle and conditional probability methods, `http://leeds-faculty.colorado.edu/laguna/articles/tabucycle.html`, 2005.

[11] Glover, F., Tabu search, Part II, *ORSA J. Comput.*, 2, 4, 1990.

[12] Laguna, M., Barnes, J. W., and Glover, F., Intelligent scheduling with tabu search: an application to jobs with linear delay penalties and sequence dependent setup costs and times, *J. Appl. Intell.*, 3, 159, 1993.

[13] Glover, F. and Laguna, M., Target analysis to improve a tabu search method for machine scheduling, *The Arabian J. Sci. Eng.*, 16, 239, 1991.

[14] Laguna, M. and Glover, F., Integrating target analysis and tabu search for improved scheduling systems, *Expert Syst. Appl.*, 6, 287, 1993.

[15] LOLIB, `http://www.iwr.uni-heildelberg.de/iwr/comopt/soft/LOLIB/LOLIB.html`, 1997.

24
Evolutionary Computation

Guillermo Leguizamón
National University of San Luis

Christian Blum
Technical University of Catalonia

Enrique Alba
University of Málaga

24.1 Introduction

This chapter aims to give a summary of evolutionary computation (EC) techniques. The summary includes a general description of the main families of algorithms belonging to the EC field as well as their main components. Also, we describe their evolution through the last years including latest advances in constraint-handling methods, parallel models and algorithms, methods for dynamic environments and multiobjective optimization, etc. Also included are some observations about the relationship between EC techniques and other methods for optimization, in particular metaheuristics. Finally, we describe some new trends and give a glimpse of the numerous applications of EC techniques.

Optimization problems are of high importance in industry and science. Examples of practical optimization problems include train scheduling, time tabling, shape optimization, or telecommunication network design. An optimization problem $\mathcal{P}$ can be described as a triple $(\mathcal{S}, \Omega, f)$, where

1. $\mathcal{S}$ is the search space defined over a finite set of decision variables X_i, $i = 1, \ldots, n$. In case these variables have discrete domains we deal with discrete optimization (or combinatorial optimization), and in case of continuous domains $\mathcal{P}$ is called a continuous optimization problem. Mixed variable problems also exist. Ω is a set of constraints among the variables.
2. $f : \mathcal{S} \rightarrow I\!R^+$ is the objective function that assigns a positive cost value to each element (or solution) of $\mathcal{S}$.

The goal is to find a solution $s \in \mathcal{S}$ such that $f(s) \leq f(s'), \forall s' \in \mathcal{S}$ (in case we want to minimize the objective function), or $f(s) \geq f(s'), \forall s' \in \mathcal{S}$ (in case the objective function must be maximized). In real-life problems the goal is often to optimize several objective functions at the same time. This form of optimization is labelled as multiobjective optimization.

Due to the practical importance of optimization problems, many algorithms to tackle them have been developed. These algorithms can be classified as either *complete* or *approximate* algorithms. Complete algorithms are guaranteed to find for every finite size instance of an optimization problem an optimal solution in bounded time (see Refs. [1,2]). Yet, many practically relevant optimization problems are—because of their size or their structure—very hard to be solved to optimality. Therefore, complete methods might need a computation time too high for practical purposes. Thus, the development of approximate methods—in which we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time—has received more and more attention in the last 30 years.

A very successful strand of approximate optimization algorithms originates from a field known as EC. EC can be regarded as a metaphor for building, applying, and studying algorithms based on Darwinian principles of natural selection. The instances of algorithms that are based on evolutionary principles are called evolutionary algorithms (EAs) [3]. Evolutionary algorithms can be characterized as computational models of evolutionary processes. They are inspired by nature's capability to evolve living beings well adapted to their environment. At the core of each EA is a population of individuals. At each algorithm iteration a number of *reproduction* operators is applied to the individuals of the current population to generate the individuals of the population of the next generation. Evolutionary algorithms might use operators called *recombination* or *crossover* to recombine two or more individuals to produce new individuals. They also might use *mutation* or *modification* operators that cause a self-adaptation of individuals. The driving force in EAs is the *selection* of individuals based on their *fitness* (which might be based on the objective function, the result of a simulation experiment, or some other kind of quality measure). Individuals with a higher fitness have a higher probability to be chosen as members of the population of the next generation (or as parents for the generation of new individuals). This corresponds to the principle of *survival of the fittest* in natural evolution. It is the capability of nature to adapt itself to a changing environment, which gave the inspiration for EAs.

There has been a variety of slightly different EAs proposed over the years. Three different strands of EAs were developed independently of each other over time. These are evolutionary programming (EP) as introduced by Fogel [4] and Fogel et al. [5], evolutionary strategies (ES) proposed by Rechenberg [6] and genetic algorithms (GAs) initiated by Holland [7] (see Refs. [8–11] for further references). Evolutionary programming arose from the desire to generate machine intelligence. While EP originally was proposed to operate on discrete representations of finite-state machines, most of the present variants are used for continuous optimization problems. The latter also holds for most present variants of ES, whereas GAs are mainly applied to solve discrete problems. More recently, other members of the EA family such as, for example, genetic programming (GP) and scatter search (SS) were developed. Despite this division into different strands, EAs can be understood from a unified point of view with respect to their main components and the way they explore the search space. Over the years there have been quite a few overviews and surveys about EC methods. Among those are the ones by Bäck [12], Fogel et al. [13], Hertz and Kobler [14], Spears et al. [15], and Michalewicz and Michalewicz [16]. In Ref. [17] a taxonomy of EAs is proposed.

The structure of this chapter is as follows. Section 24.2 gives an introduction to EAs and covers shortly important aspects such as the solution representations and search operators. Section 24.3 analyzes the evolution of EAs emphasizing some of the important recent research advances. Section 24.4 discusses and highlights the connections between EAs and other stochastic search methods, namely metaheuristics. We conclude in Section 24.5 with an overview on EAs applications and an outline of new research trends.

24.2 Evolutionary Algorithms

The algorithm in Figure 24.1 shows the basic structure of EAs. In this algorithm, P denotes the population of individuals. These individuals are not necessarily solutions to the considered problem. They may be partial solutions, or sets of solutions, or any object which can be transformed into one or more solutions in a structured way. The set of all possible individuals is generally called the *genotype space* denoted by $\mathcal{G}$, whereas the search space of the tackled optimization problem is called the *phenotype space* denoted by $\mathcal{S}$. Hereby, the structure of the individuals is called the *solution representation, or solution encoding.* $\mathcal{G}$ and $\mathcal{S}$, respectively, constitute the domain and codomain of a function g known as the *growth* (or *expression*) function. In some cases $\mathcal{G}$ and $\mathcal{S}$ are actually equivalent, g being a trivial identity function. However, this is not the general situation. As a matter of fact, the only requirement posed on g is surjectivity, that is, for each $s \in \mathcal{S}$ there must be an individual $i \in \mathcal{G}$ such that $g(i) = s$ (see Figure 24.2). Furthermore, g might be undefined for some elements in $\mathcal{G}$ or even might not be injective at all (a redundant representation).

At the start of an EA, an initial population of individuals is generated by applying a function ι to the genotype space $\mathcal{G}$. Function ι might represent a random procedure that generates individuals at random,

```
Evolutionary algorithm:
  1. $P \leftarrow$ apply $\iota$ on $\mathcal{G}$ to generate $\mu$ individuals (the initial population);
  2. while termination criteria not met do
       (a) $P' \leftarrow$ apply $\sigma$ on $P$; /* selection */
       (b) $P'' \leftarrow$ apply $\omega_r$ on $P'$; $r \in \{1, \ldots, \#operators\}$; /* reproduction */
       (c) $P \leftarrow$ apply $\psi$ on $P$ and $P''$; /* replacement */
     endwhile
```

FIGURE 24.1 Pseudocode of an evolutionary algorithm.

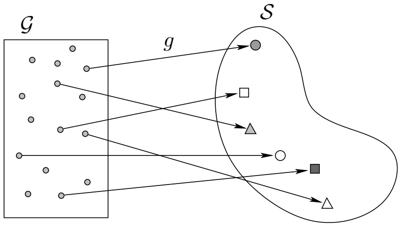


FIGURE 24.2 Mapping from the genotype space to the phenotype space.

or it might represent a heuristic seeding procedure. Then, at each iteration (also called generation) of the algorithm, the following three major operations are performed. First, a set of individuals P' are selected from the current population P by applying a function σ to P. The selection process is based on the individuals' *fitness* value, which is a measure of how good the solution represented by an individual is for the problem being considered. The fitness value is in general tightly coupled to the objective function. It is the main source for guiding the search process. Second, a population P'' of offspring is generated from P' by the application of *reproduction* operators, that is, the application of a function ω_r to P'. Finally, the replacement function ψ is applied to the current population P and the set of offspring individuals P'' to generate the population for the next generation. To choose the individuals for the next population exclusively from the offspring is called *generational replacement*. In some schemes, such as *elitist strategies*, successive generations overlap to some degree, that is, some portion of the previous generation is retained in the new population. The fraction of new individuals at each generation is called the *generational gap* [18]. In *steady-state* selection, only a few individuals are replaced in each iteration: usually a small number of the least fit individuals are replaced by offspring.

The process described above is repeated until certain termination criteria (usually reaching a maximum number of iterations, or reaching a time limit) are satisfied. Most EAs deal with populations of constant size. However, it is also possible to have a variable population size. In case of a continuously shrinking population size, the situation in which only one individual is left in the population (or no crossover partners can be found for any member of the population) might also be one of the stopping conditions of the algorithm. For a graphical presentation of an algorithm iteration see Figure 24.3.

Every possible instantiation of this general framework gives rise to different EAs. In fact, it is possible to distinguish among different EA families by considering some guidelines on how to perform this instantiation. However, before we outline the main EA families, we first deal in more detail with some important aspects that have to be taken into account when applying EAs. Figure 24.4 shows a typical line of decisions with which a practitioner is faced when applying an EA.[1] The first step consists in finding an appropriate *problem formulation*, that is, a concise and manageable description of the problem to be

[1] Note that this is similar when applying an approximate optimization technique other than EC.

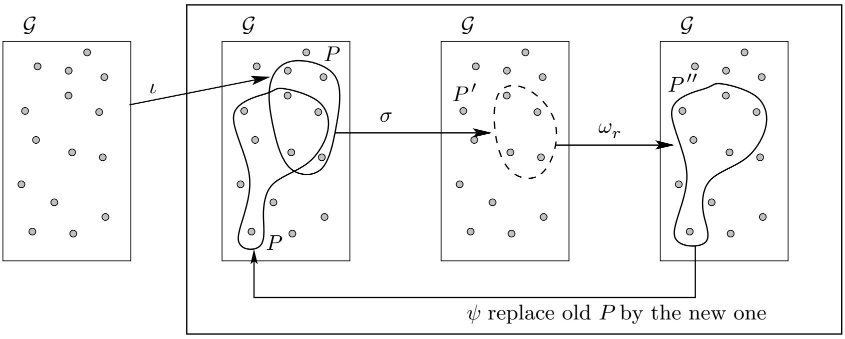


FIGURE 24.3 Illustration of the evolutionary process from the perspective of the genotype space $\mathcal{G}$.

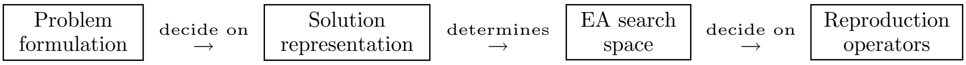


FIGURE 24.4 The different steps in the process of applying an EA to an optimization problem.

tackled (e.g., a sentence, a mathematical formula, or a mathematical program). Having decided on the problem formulation, the choice of the solution representation (which defines the EA search space), and the development of the reproduction operators are crucial for the success of the algorithm.

Solution Representation (i.e., the Individuals)

Most commonly used in EAs is the representation of solutions as bit strings or as permutations of n integer numbers. However, also real-value encodings, tree structures or other complex structures are possible. See Figure 24.5 for some examples. In GAs, an individual usually consists of one or more so-called chromosomes. Chromosomes are strings of smaller units termed genes. The different values a gene can take are called the alleles for that gene. Holland's schema analysis [7] and Radcliffe's generalization to formae [19] are examples of how theory can help to guide representation choices.

Reproduction Operators

The chosen solution encoding determines the complexity and size of the EA search space. To explore this search space it is mandatory to design a set of reproduction operators (the number and type of these operators will depend on the solution encoding and the particular EA instance). In general, we distinguish between *recombination* operators (i.e., N-ary operators) and *mutation* operators (i.e., unary operators). In most cases it is possible to recombine all individuals with each other. However, sometimes this is not the case. In general, a neighborhood function $\mathcal{N}_{\mathcal{EC}} : \mathcal{G} \rightarrow 2^{\mathcal{G}}$ assigns to each individual $i \in \mathcal{G}$ a set of individuals $\mathcal{N}_{\mathcal{EC}}(i) \subseteq \mathcal{G}$ whose members are permitted to act as recombination partners for i to create offspring. If an individual can be recombined with any other individual (as, e.g., in the simple GA [11]) we talk about *unstructured* populations, otherwise we talk about *structured* populations. An example of an EA that works on structured populations is the parallel genetic algorithm (PGA) proposed by Mühlenbein [20] (see Section 24.2.1 for more information on parallel models and algorithms).

The most common form of a recombination operator is two-parent crossover, in which one or two offspring are produced from two parents. But there are also recombination operators that operate on more than two individuals to create new individuals (multiparent crossover), (see Ref. [21,22]). More recent developments even use population statistics for generating the individuals of the next generation. Examples are the recombination operators called gene pool recombination [23] and bit-simulated crossover [24], which make use of a probability distribution over the search space given by the current population to generate the next population. The simplest form of a mutation operator just performs a small random perturbation of an individual, introducing a kind of *noise*. More complex mutation

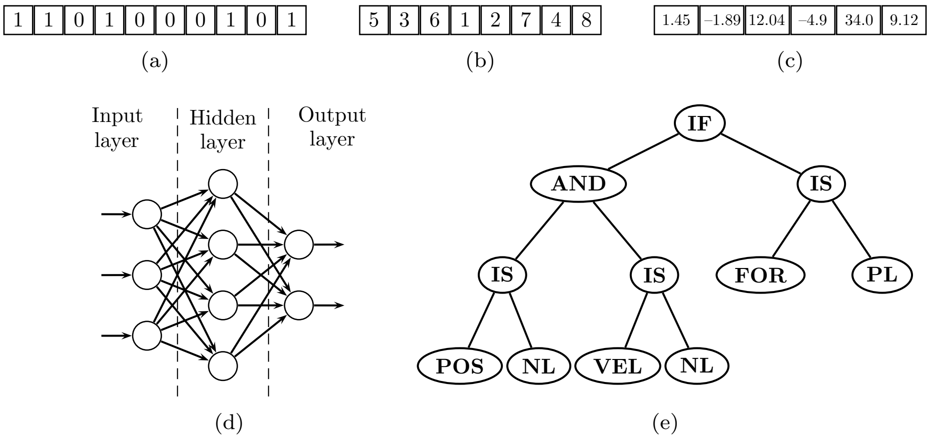


FIGURE 24.5 Five examples for solution representations. (a) Binary strings representation. They are popular, for example, to encode solutions to subset problems such as the knapsack problem. (b) Permutation representation. It is often used for permutation problems such as the traveling salesman problem. (c) The gray value (or real vector) representation. It is used, for example, for continuous optimization. (d) Graphs or trees representing a neural network. They are often represented by edge sets. (e) Trees representing a fuzzy rule. They are often used as encoding for logical rules.

operators include, for example, random moves or more goal-directed moves in a neighborhood of the individual.

It is important to remark that each possible operator will define its own characteristic neighborhood structure, i.e., the set of individuals that might be the outcome of the operators' application. Figure 24.6 and Figure 24.7 show a general view of the neighborhood sizes with respect to the whole EA search space $\mathcal{G}$. In Figure 24.6 we can observe two different neighborhoods for the same individual. For example, consider $\mathcal{G}$ to consist of all binary strings of length n. A mutation operator might be allowed to flip maximally $n/2$ bits of each individual, or, for example, only $n/10$ bits. In the first case the neighborhood of the operator is much bigger than in the second case. Therefore, different mutation operator will define neighborhoods of different sizes and complexities.

Figure 24.7 shows the neighborhood of a hypothetical (binary) recombination operator. As the distance between two individuals gets smaller (from (a) over (b) to (c)) the size of the respective neighborhood for the same operator also diminishes. The neighbor size generally depends on the operator characteristics in addition to the distance between the parent individuals. The sequence from left to right can also be

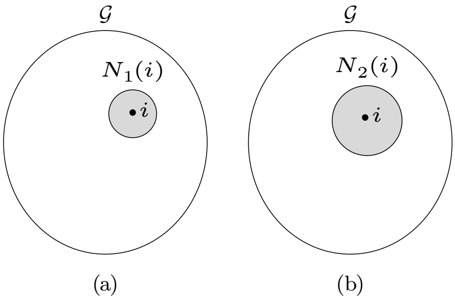


FIGURE 24.6 Graphical representation of the neighborhood $N(i)$ of a mutation operator around an individual i. The neighborhood size depends on the operator characteristics. (a) shows the example of a mutation operator with a smaller neighborhood than the one shown in (b).

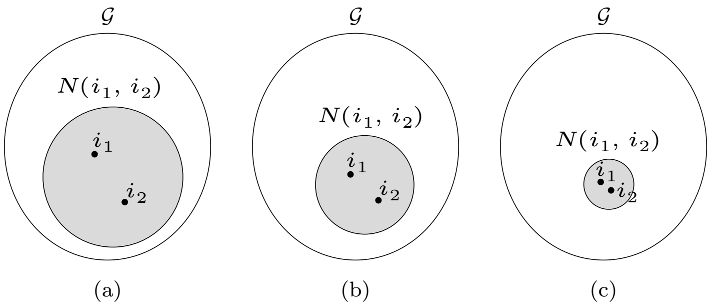


FIGURE 24.7 Graphical representation of the neighborhood $N(i_1, i_2)$ of a binary recombination operator when applied to two individuals i_1 and i_2. From (a) over (b) to (c) the distance between the two individuals changes, and therefore also the neighborhood size.

seen as the changing exploration capacity of the operator as the population tends to converge, that is, the individuals from the population are getting closer, which means that the EA converges.

Finally, in Table 24.1, we provide some commonly used reproduction operators for popular solution representations.

Exploitation versus Exploration

Of great importance for the success of an EA is the right balance between *exploitation* and *exploration*. The term exploitation refers to the use of the accumulated search experience, whereas the term exploration generally refers to the search for areas in the search space that were not visited yet. In terms of exploitation, it proved in many EA applications quite beneficial to use improvement mechanisms to increase the fitness of individuals. While the use of a population ensures an exploration of the search space, the use of local search techniques helps to quickly identify "good" areas in the search space. Another exploitation strategy is the use of recombination operators that explicitly try to combine "good" parts of individuals (rather than, e.g., a simple one-point crossover for bit strings). This may guide the search performed by EAs to areas of individuals with certain "good" properties. Techniques of this kind are sometimes called *linkage learning* or *building block learning* (see, e.g., Refs. [25–28]). Furthermore, generalized recombination operators which incorporate the notion of "neighborhood search" into EC have been proposed in the literature. An example can be found in Ref. [29].

One of the major difficulties of EAs (especially when applying local search) is the premature convergence toward suboptimal solutions. The simplest mechanism to increase exploration is the use of a mutation operator. The simplest form of a mutation operator just performs a small random perturbation of an individual, introducing a kind of *noise*. To avoid premature convergence there are also a number of other ways of maintaining the population diversity. Probably the oldest strategies are *crowding* [18] and its close relative, *preselection* [30]. Newer strategies are *fitness sharing* [31], (respectively *niching* [32]), which is a

TABLE 24.1 Some Representative Solution Encodings together with Commonly Used Reproduction Operators

Solution Representation	**Reproduction Operators**
Binary strings	One-point crossover, uniform crossover, flip mutation, etc.
Permutations	Partially matched crossover (PMX), order crossover (OX), SWAP mutation, etc.
Gray value encoding (i.e., real vectors)	Arithmetic crossover, *k*-point crossover, Gauss mutation, etc.
Integer vectors	*k*-point crossover, little and big creep mutation, random mutation, etc.
S-expressions	Subree exchange, node function alteration, etc.

collective name, whereby the reproductive fitness allocated to an individual in a population is reduced proportionally to the number of other individuals that share the same region of the search space.

24.2.1 Families of Evolutionary Algorithms

Evolutionary algorithms, as we know them today, were first introduced during the late 1960s and early 1970s. In these years, scientists from different places in the world almost simultaneously began to transfer nature's optimization capabilities into algorithms for search and problem solving. The existence of these different primordial sources resulted in the rise of three different EA models. These classical families are:

- *Evolutionary programming*. This EA family originated from the work of Fogel et al. [5]. EP focuses on the adaption of individuals rather than on the evolution of their genetic information. This implies a much more abstract view of the evolutionary process, in which the behavior of individuals is directly modified (as opposed to manipulating its genes). This behavior is typically modeled by using complex data structures such as finite automata or graphs (see, e.g., Figure 24.5 [d]). Traditionally, EP uses asexual reproduction (i.e., mutation) by introducing slight changes in an existing solution, and selection techniques based on direct competition among individuals.
- *Evolutionary strategies*. These techniques were initially developed by Rechenberg [6]. Their original goal was to create a tool for solving engineering problems. With this goal in mind, these techniques are characterized by manipulating arrays of floating-point numbers (see, e.g., Figure 24.5 [c]). Nowadays also exist versions of ES for discrete problems, but still their application to continuous optimization problems is predominant. As in EP, mutation is often the unique reproductive operator used in ES; however, sometimes also recombination operators are used within ES. A very important feature of ES is the utilization of self-adaptive mechanisms for controlling the application of mutation. These mechanisms are aimed at optimizing the progress of the search by evolving not only the individuals, but also some parameters for mutating these individuals (in a typical situation, an ES individual is a pair $(i, \vec{\sigma})$, where $\vec{\sigma}$ is a vector of standard deviations used to control the Gaussian mutation exerted on the individual i).
- *Genetic algorithms*. These are possibly the most widespread variant of EAs. They were first introduced by Holland [7]. Holland's work had a great influence on the developments in the field, to the point that some aspects of his work nearly achieved dogma status (e.g., the ubiquitous use of binary strings as solution representation, as shown, for example, in Figure 24.5 [a]). Later, the seminal book of Goldberg [8] resulted in a widespread use of GAs for discrete optimization. The main feature of GAs is the use of a recombination (or *crossover*) operator as the primary search tool. The rationale is the assumption that different parts of the optimal solution can be independently discovered, and be later combined to create better solutions. Additionally, mutation is used, but is generally considered a secondary background operator whose purpose is merely "to keep the pot boiling" by introducing new information in the population. However, this classical interpretation is no longer considered valid nowadays.

The three EA families described above have not developed in complete isolation from each other. On the contrary, numerous researchers have built bridges among them, and cross-fertilization was and is common. As a result of this interaction, the borders of these classical families tend to be fuzzy (the reader may consult [12] for a unified presentation of EA families), and new variants have emerged of which we cite the following ones:

- *Evolution programs*. This term owes to Michalewicz [33], and comprises those techniques that, while using the fundamental principles of GAs, evolve complex data structures, as in EP. Nowadays, it is accepted to use the acronym GA—or more generally EA—to refer to such an algorithm, leaving the term "traditional GA" to denote classical bit-string-based GAs.
- *Genetic programming*. The roots of GP can be traced back to the work of Cramer [34]. However, nowadays this technique is mostly associated with Koza, who promoted GP to its current status [35].

Essentially, GP could be viewed as an evolution program in which the structures evolved represent computer programs. Such programs are typically encoded by trees (see, e.g., Figure 24.5 [e]). The ultimate goal of GP is the automatic design of a program for performing a certain task, formulated as a collection of (input, output) examples.

- *Memetic algorithms (MA).* These techniques owe their name to Moscato and Cotta [36]. Some widespread misconception equates MAs to EAs augmented with local search; although such an augmented EA could be indeed considered an MA, there are additional properties that define MAs (e.g., restarting procedures). In general, an MA is a problem-aware EA. This problem awareness is typically acquired by combining the EA with existing algorithms such as hill climbing, branch and bound, etc.
- *Scatter search.* The original ideas for SS go back to Glover [37] and originated from strategies for creating composite decision rules and surrogate constraints [38]. The generation of new solutions is achieved by systematically following unifying principles for joining solutions based on generalized path constructions in Euclidean spaces. The terminology in SS is slightly different to the rest of the EAs. For example, the main population of individuals is called the *reference set.* This reference set is manipulated by the application of the following five methods: (1) diversification generation, (2) improvement, (3) reference set update, (4) subset generation, and (5) solution combination.

24.3 Advanced EA Research Topics

Due to their success in practice, EAs have received an evergrowing amount of attention during the last two decades. Nowadays, the diversity of research on EAs is impressive. Besides the improvement of their theoretical understanding, the main aims of this research concern their efficiency and the discovery of new problem domains to which they can be applied. More and more advanced versions of EAs for increasingly complex problems were developed. In the following we will shortly deal with the following examples: EAs for constrained optimization problems (COPs), for dynamic optimization problems (DOPs), and for multiobjective optimization problems (MOPs). Figure 24.8 shows a time line which—in addition to the main EA families—indicates the start of the research activities in these areas. Furthermore, we will shortly present parallel EAs (PEAs) whose aim is to exploit the implicit parallelism that is based on the fact that EAs are population-based techniques.

Constrained Problems

So far we have assumed that the outcome of a reproduction operator consists always of one ore more feasible individuals. However, sometimes reproduction operators are applied that do not guarantee that. This is the case in particular when real-world problems are concerned that often include constraints in their formulation. However, early applications of EAs were designed to solve mainly unconstrained

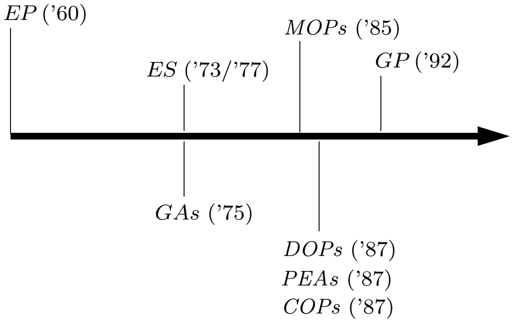


FIGURE 24.8 Time line of EA research including the main EA families and some new developments. The dates indicate the beginning of the respective research activity.

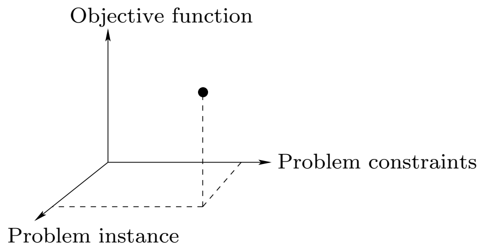


FIGURE 24.9 Dynamic environments seen as a three-dimensional field.

combinatorial and continuous optimization problems. As soon as the research community began to explore the application of EAs to more realistic problems, several constraint handling techniques were accordingly developed. The simplest method is to *reject* infeasible individuals. Nevertheless, for some highly constrained problems (e.g., for timetabling problems) it might be very difficult to find feasible individuals. A very popular method is the incorporation of constraints into the fitness function through a penalty factor. However, this technique experiences many drawbacks due to the need of finding appropriate penalty factors. Therefore, other proposals were designed to automatically adapt the penalty factors during the evolutionary process. In addition, plenty of works have been devoted in the last years to develop improved and alternative constraint-handling techniques including advanced penalty approaches, repair of infeasible individuals, design of special representations and operators, and several versions of hybrid methods. For a comprehensive survey of constraint-handling techniques see Ref. [39].

Dynamic Problems

One of the most recent applications of EAs is the application to DOPs. These are problems that exhibit changes in the problem formulation during the algorithms' execution. These changes might concern different parts of the problem formulation. For example, Morrison [40] exclusively dealt with changes of the objective function. These problems are also called problems with a *dynamic fitness landscape*. A different example is the work of Branke [41], who adopts a more general point of view in which the problem might change either in the objective function, in the problem instance, or in terms of the problem constraints (see Figure 24.9). A general definition of DOPs does not exist so far. However, Branke gives a categorization of dynamic environments with respect to the frequency, severity, predictability, and cycle of the changes [41]. The central aspects of EAs that are specially designed for dynamic optimization are the introduction and maintainance of diversity in the population, and the use of implicitly or explicitly stored search history. These adaptive EAs must find a trade-off between the quality, robustness, and flexibility of the found solutions.

Multiobjective Problems

Multiobjective optimization is another important research field in which EAs have been successfully applied [42]. As a population-based technique, EAs seem to be particularly suitable to face this class of problems. In contrast to other traditional techniques, EAs provide an entire set of Pareto optimal solutions in a single run. During the last years a number of different EA-based approaches have been proposed to deal with multiobjective optimization. Some of the most popular ones are (according to Coello [42]): the aggregation of the objective functions (combinations of all objectives into a single one), VEGA[2] (an extended version of Grefenstette's GENESIS that implements an alternative selection operator), MOGA (is a scheme in which the population is ranked on the basis of nondominance; this rank is used to accordingly calculate each individuals' fitness value), NSGA (bases its behavior on successive classification and manipulation of nondominated individuals; this allows to search and quickly converge to nondominated regions), and

[2] VEGA was probably the first approach proposed by Schaffer [43] explicitly aimed at solving MOPs.

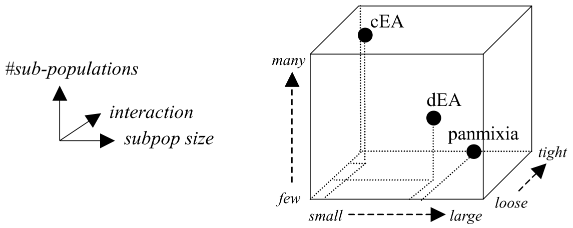


FIGURE 24.10 The structured-population genetic algorithm cube.

NPGA (implements an extended version of tournament selection in which the competitors compete against a certain number of other individuals). Of course, each of these approaches has strengths and weakness. However, they offer a clear evidence of the intrinsic flexibility and robustness of EAs which makes them suitable for tackling MOPs.

Parallel Models and Algorithms

Even though the use of populations in EAs has many advantages, EAs often suffer from the disadvantage of being highly computation time consuming. This is especially the case when the application of the fitness function is costly, or when the population size is required to be large. However, EAs are naturally prone to parallelism since the most usual operations on the individuals are mostly independent from each other. Besides that, the whole population (also called panmixia) can be geographically structured in separate subpopulations, often leading to better algorithms. The geographical structuring of a population is a form of decentralization. Decentralizing an EA by structuring the population has many advantages, but using structured algorithms may have some disadvantages too, as for example, the higher complexity of implementation and analysis, added to the fact that decentralizing the algorithm is not always better. Research on EAs that exploit these aspects, called PEAs [44], give often evidence of higher efficiency, larger diversity maintenance, higher availability of memory and CPU resources, and multisolution capabilities of these algorithms. In the following, we shortly describe two important models of PEAs.

- In coarse-grained or distributed EAs (dEAs) the population is partitioned into several subpopulations (islands). Each island is treated independently, and there are exchanges of information among them.
- In fine-grained or cellular evolutionary algorithms (cEAs) the individuals are placed on a toroidal N-dimensional grid (where $N = 1, 2, 3$ is used in practice), with one individual per grid location (this location is often referred to as a cell, the fine-grained approach being also known as cellular). Every individual has a neighborhood and can only be recombined with individuals from this neighborhood. The main difference of a cEA with respect to a panmistic EA is its decentralized selection, since the reproductive loop is performed inside each of the numerous pools. Hereby, each individual has its own pool composed of neighboring individuals, and at the same time, this individual belongs to many other pools. Usually, a two-dimensional structure with overlapped neighborhoods is used to provide a smooth diffusion of good solutions along the grid.

The relation between dEAs and cEAs is graphically represented in Figure 24.10. Structured EAs usually outperform standard EAs (i.e., unstructured EAs) numerically. A way for further improving PEAs lies in parallelized implementations and further decentralization.

24.4 EAs in the Context of Metaheuristics

In a wider context, EAs belong to the class of metaheuristic algorithms, which are search algorithms for approximate optimization. The term *metaheuristic*, first introduced in Ref. [45], derives from the composition of two Greek words. *Heuristic* is derived from the verb *heuriskein* ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$) which means "to find", while the suffix *meta* means "beyond, in an upper level." Before this term was widely adopted, metaheuristics were often called *modern heuristics* [46]. Except for EAs, this class of algorithms includes[3]—but is not restricted to—ant colony optimization (ACO), iterated local search (ILS), simulated annealing (SA), and tabu search (TS).

Metaheuristics can be characterized as high-level strategies for exploring search spaces by defining general rules that are later customized to the problem at hands. Of great importance hereby is that a dynamic balance is achieved between *diversification* and *intensification*. The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience. These terms stem from the TS field [47] and it is important to clarify that the terms *exploration* and *exploitation* are used instead in EC [48] (as outlined before). The balance between diversification and intensification is important, on one side to quickly identify regions in the search space with high-quality solutions, and on the other side not to waste too much time in regions of the search space which are either already explored or which do not provide high-quality solutions. Blum and Roli [49] elaborated on the importance of the two concepts in their survey on metaheuristics.

The search strategies of different metaheuristics are highly dependent on the philosophy of the metaheuristic itself. There are several different philosophies apparent in the existing metaheuristics. Some of them can be seen as "intelligent" extensions of local search algorithms. The goal of this kind of metaheuristic is to escape from local minima to proceed in the exploration of the search space and to move on to find other hopefully better local minima. This is, for example, the case in TS, ILS, VNS, and SA. These metaheuristics (also called trajectory methods) work on one or several neighborhood structure(s) imposed on the search space. We can find a different philosophy in algorithms such as ACO and EAs. They incorporate a learning component in the sense that they implicitly or explicitly try to learn correlations between decision variables to identify high-quality areas in the search space. This kind of metaheuristic performs, in a sense, a biased sampling of the search space. For instance, in EAs this is achieved by recombination of solutions and in ACO by sampling the search space at each iteration according to a probability distribution.

An important research direction is the hybridization of EAs with other techniques for optimization, and in particular with other metaheuristics. One of the most popular ways of hybridization concerns the use of trajectory methods in EAs. Many of the successful applications of EC make use of this feature. The reason for that becomes apparent when analyzing the respective strengths of trajectory methods and EAs. The power of EAs is certainly based on the concept of recombining solutions to obtain new ones. This allows to make guided steps in the search space which are usually "larger" than the steps done by trajectory methods. In other words, a solution resulting from a recombination in EAs is usually more "different" from the parents than, say, a predecessor solution to a successor solution (obtained by applying a move) in TS. We also have "large" steps in trajectory methods such as ILS and VNS, but in these methods the steps are usually not guided (these steps are rather called "kick move" or "perturbation" indicating the lack of guidance). It is interesting to note that in population-based methods there are mechanisms in which elite solutions from the search history influence the search process in the hope of finding better solutions in-between those elite solutions and the current solutions. In EAs this is often obtained by keeping the best (or a number of the best) solution(s) found since the beginning of the respective run of the algorithm in the population. This is called a *steady state* evolution process. Scatter search performs a steady-state process by definition. In contrast, the strength of trajectory methods is rather to be found in the more structured way in which they explore a promising region in the search space. In this way the danger of being close to good solutions but "missing" them is not as high as in population-based methods such as EAs.

In summary, population-based methods are better in identifying promising areas in the search space, whereas trajectory methods are better in exploring promising areas in the search space. Thus, EA hybrids that in some way manage to combine the advantage of population-based methods with the strength of trajectory methods are often very successful.

[3] In alphabetical order.

24.5 Applications and New Trends

Evolutionary algorithms have been applied to most optimization problems (discrete, continuous, as well as multiobjective). In the classical area of combinatorial optimization, for example, EAs have been applied to NP-hard problems such as the traveling salesman problem, the multiple knapsack problem, number partitioning, max independent set, and graph coloring, among others. Other nonclassical–yet important–COPs to which EAs have been applied are scheduling (in many variants), timetabling, vehicle routing, quadratic assignment, placement problems, and transportation problems. Telecommunications is also a field that has witnessed successful applications of EAs. For example, EAs have been applied to the placement of antennas and converters, frequency assignment, digital data network design, predicting bandwidth demands in ATM networks, error code design, etc. Evolutionary algorithms have been actively used in electronics and engineering as well. For example, work has been done in structure optimization, aeronautic design, power planning, circuit design, computer-aided design, analogue network synthesis, and service restoration, among other areas. For an extensive collection of references to EC applications we refer to Ref. [3].

In addition to the above mentioned areas of applications there is an increasing interest in new areas for which EAs seems to be a good choice. Recent successes were obtained, for example, in the rapidly growing bioinformatics area (see, e.g., Refs. [50–52]), software engineering [53], the banking sector [54,55], computer imagery [56], drug design [57], clustering [58], cryptology [59], games [60,61], elevator groups [62], and evolvable hardware [63].

The research field of EC is vast, indeed, and continuously under development. This is indicated by a significant number of international conferences such as

- the *International Congress on Evolutionary Computation (CEC)*;
- the *Genetic and Evolutionary Computation Conference (GECCO)*;
- the *European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP)*;
- the *Metaheuristics International Conference (MIC)*; and
- the *International Conference on Parallel Problems Solving in Nature (PPSN)*, among others.

Futhermore exist a number of periodical journals such as *Evolutionary Computation, and IEEE Transactions on Evolutionary Computation,* etc., which are devoted specifically to this area. The interested reader may also want to query any bibliographical database or web search engine for "evolutionary algorithm application" to get an idea of the vast number of problems that have been tackled with EAs.

24.6 Conclusions

In this chapter we gave an introduction into the working of EAs. We presented an overview of the main families of EAs as well as a description of their main components from a unifying perspective. Furthermore we dealt with advanced research directions in EC such as parallel models and algorithms, and the application of EAS to constrained, dynamic, and multiobjective problems. We have put EAs into perspective by describing them as a member of the class of metaheuristic algorithms, highlighting hereby the strengths of EAs in contrast to the strengths of other metaheuristics. Finally, we briefly dealt with typical as well as with more recent EA applications. The rich diversity of EA applications gives a flavor of the popularity and the generality of these techniques.

Acknowledgments

This work has been partially funded by the Ministry of Science and Technology (MCYT) and Regional Development European Found (FEDER) under contract TIC2002-04498-C05-02 (the TRACER project) `"http://tracer.lcc.uma.es"`. Christian Blum acknowledges support from the *Ramóy Cajal* program of the Spanish Ministry of Science and Technology.

References

[1] Nemhauser, G. L. and Wolsey, A. L., *Integer and Combinatorial Optimization*, Wiley, New York, 1988.

[2] Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization—Algorithms and Complexity*, Dover Publications, Inc., Mineola, NY, 1982.

[3] Bäck, T., Fogel, D. B., and Machalewicz, Z., Eds., *Handbook of Evolutionary Computation*, Institute of Physics Publishing Ltd, Bristol and Philadelphia, UK, 2000.

[4] Fogel, L. J., Toward inductive inference automata, *Proc. Int. Federation for Information Processing Congress*, 1962, p. 395.

[5] Fogel, L. J., Owens, A. J., and Walsh, M. J., *Artificial Intelligence through Simulated Evolution.* Wiley, New York, 1966.

[6] Rechenberg, I., *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog, Stuttgart, Germany, 1973.

[7] Holland, J. H., *Adaption in Natural and Artificial Systems*, The University of Michigan Press, Ann Harbor, MI, 1975.

[8] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[9] Mitchell, M., *An Introduction to Genetic Algorithms*, MIT press, Cambridge, MA, 1998.

[10] Reeves, C. R. and Rowe, J. E., *Genetic Algorithms: Principles and Perspectives. A Guide to GA Theory*, Kluwer Academic Publishers, Boston, MA, 2002.

[11] Vose, M. D., *The Simple Genetic Algorithm: Foundations and Theory*, MIT Press, Cambridge, MA, 1999.

[12] Bäck, T., *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, 1996.

[13] Fogel, D. B., An introduction to simulated evolutionary optimization, *IEEE Trans. Neural Networks*, 5(1), 3, 1994.

[14] Hertz, A. and Kobler, D., A framework for the description of evolutionary algorithms, *Eur. J. Oper. Res.*, 126, 1, 2000.

[15] Spears, W. M., De Jong, K. A., Bäck, T., Fogel, D. B., and de Garis, H., An overview of evolutionary computation, in *Proceedings of the European Conference on Machine Learning (ECML-93)*, Vol. 667, Brazdil, P. B., Ed., Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 1993, p. 442.

[16] Michalewicz, Z. and Michalewicz, M., Evolutionary computation techniques and their applications, *Proc. IEEE Int. Conf. on Intelligent Processing Systems*, Beijing, China, 1997, p. 14.

[17] Calégary, P., Coray, G., Hertz, A., Kobler, D., and Kuonen, P., A taxonomy of evolutionary algorithms in combinatorial optimization, *J. Heuristics*, 5, 145, 1999.

[18] DeJong, K. A., An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. thesis, University of Michigan, Ann Arbor, MI, 1975. Dissertation Abstracts International 36(10), 5140B, University Microfilms Number 76-9381.

[19] Radcliffe, N. J., Forma analysis and random respectful recombination, *Proc. 4th Int. Conf. on Genetic Algorithms, ICGA 1991*, Morgan Kaufmann, San Mateo, CA, 1991, p. 222.

[20] Mühlenbein, H., Evolution in time and space—the parallel genetic algorithm, in *Foundations of Genetic Algorithms*, Rawlins, J. E., Ed., Morgan Kaufmann, San Mateo, CA, 1991.

[21] Bersini, H. and Seront, G., In search of a good evolution-optimization crossover, *Proceedings of PPSN-II, Second International Conference on Parallel Problem Solving from Nature*, Männer, R. and Manderick, B., Eds., Elsevier, Amsterdam, The Netherlands, 1992, p. 479.

[22] Eiben, A. E., Raué, P.-E., and Ruttkay, Z., Genetic algorithms with multi-parent recombination, in *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, Vol. 866, Davidor, Y., Schwefel, H.-P., and Manner, R., Eds., Lecture Notes in Computer Science, Springer, Berlin, 1994, p. 78.

[23] Mühlenbein, H. and Voigt, H.-M., Gene pool recombination in genetic algorithms, in *Proceedings of the Metaheuristics Conference*, Osman I. H. and Kelly J. P., Eds., Kluwer Academic Publishers, Dordrecht, 1995.

[24] Syswerda, G., Simulated crossover in genetic algorithms, *Proc. 2nd Workshop on Foundations of Genetic Algorithms*, Whitley, L. D., Ed., San Mateo, CA, 1993, p. 239.

[25] Goldberg, D. E., Deb, K., and Korb, B., Don't worry, be messy, *Proc. 4th Int. Conf. on Genetic Algorithms*, La Jolla, CA, 1991.

[26] Harik, G., Linkage learning via probabilistic modeling in the ECGA, Technical report No. 99010, IlliGAL, University of Illinois, 1999.

[27] van Kemenade, C. H. M., Explicit filtering of building blocks for genetic algorithms, *Proceedings 4th Conf. on Parallel Problem Solving from Nature—PPSN IV*, Vol. 1141, Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., Eds., Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 1996, p. 494.

[28] Watson, R. A., Hornby, G. S., and Pollack, J. B., Modeling building-block interdependency, in *Late Breaking Papers at the Genetic Programming 1998 Conference*, Koza, J. R., Ed., University of Wisconsin, Madison, Wisconsin, 1998.

[29] Rayward-Smith, V. J., A unified approach to tabu search, simulated annealing and genetic algorithms, in *Applications of Modern Heuristics*, Rayward-Smith, V. J., Ed., Alfred Waller Limited, Oxfordshire, UK, 1994.

[30] Cavicchio, D. J., Adaptive Search Using Simulated Evolution, Ph.D. thesis, University of Michigan, Ann Arbor, MI, 1970.

[31] Goldberg, D. E., and Richardson, J., Genetic algorithms with sharing for multimodal function optimization, in *Genetic Algorithms and their Applications*, Grefenstette, J. J., Ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, p. 41.

[32] Mahfoud, S. W., Niching Methods for Genetic Algorithms, Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1995.

[33] Michalewicz, A., *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed., Springer, Berlin, 1998.

[34] Cramer, M. L., A representation for the adaptive generation of simple sequential programs. *Proc. 1st Int. Conf. on Genetic Algorithms*, Grefenstette, J. J., Ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

[35] Koza, J. R., *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

[36] Moscato, P. and Cotta, C., A gentle introduction to memetic algorithms, in *Handbook of Metaheuristics*, Glover, F. and Kochenberger G., Eds., Kluwer Academic Publishers, Boston MA, 2003, p. 105.

[37] Glover, F., Parametric Combinations of Local Job Shop Rules, ONR Research Memorandum No. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1963.

[38] Laguna, M. and Martí, R., *Scatter Search. Methodology and Implementations in C*, Kluwer Academic Publishers, Boston, MA, 2003.

[39] Coello Coello, C. A., A comprehensive survey of evolutionary-based multiobjective optimization techniques, *Knowl. Inf. Syst.*, 1(3), 269, 1999.

[40] Morrison, R. W., *Designing Evolutionary Algorithms for Dynamic Environments*, Springer, Berlin, 2004.

[41] Branke, J., *Evolutionary Optimization in Dynamic Problems*, Kluwer Academic Publishers, Dordrecht, 2002.

[42] Coello Coello, C. A., An updated survey of GA-based multiobjective optimization techniques, *ACM Comput. Surv.*, 32(2), 109, 2000.

[43] Schaffer, J. D., Multiple objective optimization with vector evaluated genetic algorithms, in *Genetic Algorithms and Their Applications: 1st Int. Conf. on Genetic Algorithms*, Vol. 100, Grefensette, J. J., Ed., Erlbaum, Hillsdale, NJ, 1985, p. 93.

[44] Alba, E. and Tomassini, M., Parallelism and evolutionary algorithms, *IEEE Trans. Evol. Comput.*, 6(5), 443, 2002.

[45] Glover, F., Future paths for integer programming and links to artificial intelligence, *Comput. Operations Res.*, 13, 533, 1986.

[46] Reeves, C. R., Ed., *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publishing, Oxford, England, 1993.

[47] Glover, F. and Laguna, M., *Tabu Search*, Kluwer Academic Publishers, Dordrecht, 1997.

[48] Eiben, A. E. and Schippers, C. A., On evolutionary exploration and exploitation, *Fund. Informaticae*, 35, 1, 1998.

[49] Blum, C. and Roli, A., Metaheuristics in combinatorial optimization: overview and conceptual comparison, *ACM Comput. Surv.*, 35(3), 268, 2003.

[50] Fogel, G. B., Porto, V. W., Weekes, D. G., Fogel, D. B., Griffey, R. H., McNeil, J. A., Lesnik, E., Ecker, D. J., and Sampath, R., Discovery of RNA structural elements using evolutionary computation, *Nucleic Acids Res.*, 30(23), 5310, 2002.

[51] Seehuus, R., Tveit, A., and Edsberg, O., Discovering biological motifs with genetic programming: Comparing linear and tree-based representations for unaligned protein sequences, *Proc. Genetic and Evolutionary Computation Conf. (GECCO-2005)*, Vol. 1, Washington, DC, 2005, p. 401.

[52] Rowland, J., On genetic programming and knowledge discovery in transcriptome data, *Proc. Congress on Evolutionary Computation (CEC'05)*, Vol. 1, Portland, OR. 2004, p. 158.

[53] Briand, L. C., Labiche, Y., and Shousha, M., Stress testing real-time systems with genetic algorithms, *Proc. Genetic and Evolutionary Computation Conf. (GECCO-2005)*, Vol. 1, Washington, DC, 2005, p. 1021.

[54] Budynek, J., Bonabeau, E., and Shargel, B., Evolving computer intrusion scripts for vulnerability assessment and log analysis, *Proc. Genetic and Evolutionary Computation Conf. (GECCO-2005)*, Washington, DC, 2005, p. 1905.

[55] Wagner, N., Michalewicz, Z., Khouja, M., and McGregor, R. R. N., Time series forecasting for dynamic environments: the dyfor genetic program model, *IEEE Trans. Evol. Comput.*, (submitted).

[56] Grasemann, U. and Miikkulainen, R., Effective image compression using evolved wavelets, *Proc. Genetic and Evolutionary Computation Conf. (GECCO-2005)*, Vol. 2, Washington, DC, 2005, p. 1961.

[57] Lameijer, E. W., Ijzerman, A., Kok, J., and Bäck, T., The molecule evaluator: an interactive evolutionary algorithm for designing drug molecules, *Proc. Genetic and Evolutionary Computation Conf. (GECCO-2005)*, Washington, DC, 2005, p. 1969.

[58] Sheng, W. and Liu, X., A hybrid algorithm for K-medoid clustering of large data sets, *Proc. Congress on Evolutionary Computation (CEC'05)*, Vol. 1, Portland, OR, 2004, p. 77.

[59] Seredynski, M. and Bouvry, P., Block cipher base on reversible cellular automata, *Proc. Congress on Evolutionary Computation (CEC'05)*, Portland, OR, 2004, p. 2138.

[60] Denzinger, J., Chan, B., Gates, D., Loose, K., and Buchanan, J., Evolutionary behavior testing of commercial computer games, *Proc. Congress on Evolutionary Computation (CEC'04)*, Vol. 1, Portland, OR, 2004, p. 125.

[61] Fogel, D. B., Evolving strategies in blackjack, *Proc. Congress on Evolutionary Computation (CEC'04)*, Portland, OR, 2004, p. 1427.

[62] Eguchi, T., Hirasawa, K., Hu, J., and Markon, S., Elevator group supervisory control systems using genetic network programming, *Proc. Congress on Evolutionary Computation (CEC'04)*, Portland, OR, 2004, p. 1661.

[63] Sipper, M., Sanchez, E., Mange, D., Tomassini, M., Pérez-Uribe, A., and Stauffer, A., A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems, *IEEE Trans. Evol. Comput.*, 1(1), 83, 1997.

25
Simulated Annealing

Emile Aarts
Philips Research Laboratories

Jan Korst
Philips Research Laboratories

Wil Michiels
Philips Research Laboratories

25.1 Introduction

In the early 1980s, Kirkpatrick et al. [1] and, independently, Černý [2] introduced simulated annealing as a randomized local search algorithm to solve combinatorial optimization problems. In a combinatorial optimization problem we are given a finite or countably infinite set of solutions S and a cost function f that assigns a cost to each solution. The problem is to find a solution $i^* \in S$ for which $f(i^*)$ is either minimal or maximal, depending on whether the problem is a minimization or a maximization problem. Such a solution i^* is called a (globally) optimal solution. Without loss of generality, we restrict ourselves in this chapter to minimization problems.

Simulated annealing is a local search algorithm, which means that it starts with an initial solution and then searches through the solution space by iteratively generating a new solution that is "near" to the current solution. A neighborhood function $N: S \to 2^S$ defines for any given solution $s \in S$, the set $N(s) \subseteq S$ of solutions that are near to it. The set $N(s)$ is called the neighborhood of solution s, and each $s' \in N(s)$ is called a neighbor of s. The process of searching through the solution space can be modeled as a walk through a directed graph $G = (V, E)$, where node set V is given by the solution space S and arc set E contains arc (i, j) if and only if $j \in N(i)$. This graph G is called the neighborhood graph.

The solution space of combinatorial optimization problems can typically be formulated in terms of discrete structures, such as sequences, permutations, graphs, and partitions. Local search uses these representations by defining neighborhood functions in terms of local rearrangements, such as moving, swapping, and replacing items, that can be applied to a representation to obtain a neighboring solution.

The simplest form of local search is iterative improvement. An iterative improvement algorithm continuously explores neighborhoods for a solution with lower cost. If such a solution is found, then the current solution is replaced by this better solution. The procedure is repeated until no better solutions can be found in the neighborhood of the current solution. By definition, iterative improvement terminates in a local optimum, which is a solution having a cost at least as good as all of its neighbors.

A disadvantage of using iterative improvement is that it easily gets trapped in poor local optima. To avoid this disadvantage, simulated annealing accepts in a limited way neighboring solutions with a cost that is worse than the cost of the current solution.

25.2 Basic Simulated Annealing

Originally, the use of simulated annealing in combinatorial optimization was heavily inspired by an analogy between the physical annealing process of solids and the problem of solving large combinatorial optimization problems. Since this analogy is quite appealing we use it here as a background for introducing simulated annealing.

In condensed matter physics, annealing is known as a thermal process for obtaining low-energy states of a solid in a heat bath. The process consists of the following two steps [1]:

- Increase the temperature of the heat bath to a maximum value at which the solid melts.
- Decrease carefully the temperature of the heat bath until the particles arrange themselves in the ground state of the solid.

In the liquid phase all particles arrange themselves randomly, whereas in the ground state of the solid, the particles are arranged in a highly structured lattice, for which the corresponding energy is minimal. The ground state of the solid is obtained only if the maximum value of the temperature is sufficiently high and the cooling is done sufficiently slow. Otherwise the solid will be frozen into a metastable state rather than into the true ground state.

It is known that, if the lowering of the temperature is done sufficiently slow, the solid can reach thermal equilibrium at each temperature. Already in 1953, Metropolis et al. [28] introduced a simple algorithm for simulating the evolution of a solid in a heat bath to thermal equilibrium. Their algorithm is based on Monte Carlo techniques [3], and generates a sequence of states of the solid in the following way. Given a current state i of the solid with energy E_i, a subsequent state j is generated by applying a perturbation mechanism that transforms the current state into a next state by a small distortion, for instance, by a displacement of a single particle. The energy of the next state is E_j. If the energy difference, $E_j - E_i$, is less than or equal to 0, the state j is accepted as the current state. If the energy difference is greater than 0, then state j is accepted with a probability given by

$$\exp\left(\frac{E_i - E_j}{k_B T}\right)$$

where T denotes the temperature of the heat bath and k_B a physical constant known as the Boltzmann constant. The acceptance rule described above is known as the Metropolis criterion and the algorithm that goes with it is known as the Metropolis algorithm. In the Metropolis algorithm thermal equilibrium is achieved by generating a large number of transitions at a given temperature value. Thermal equilibrium is characterized by the Boltzmann distribution, which gives the probability of the solid to be in a state i with energy E_i at temperature T, and is given by

$$\mathbb{P}_T\{\mathbf{X} = i\} = \frac{\exp(-E_i/k_B T)}{\sum\limits_j \exp(-E_j/k_B T)} \tag{25.1}$$

where $\mathbf{X}$ is a random variable denoting the current state of the solid and the summation extends over all possible states. As we show below, the Boltzmann distribution plays an essential role in the analysis of the convergence of simulated annealing.

Returning to simulated annealing, the Metropolis algorithm can be used to generate a sequence of solutions of a combinatorial optimization problem by assuming the following equivalences between a physical many-particle system and a combinatorial optimization problem:

- Solutions in the combinatorial optimization problem are equivalent to states of the physical system.
- The cost of a solution is equivalent to the energy of a state.

Simulated annealing can be viewed as a sequence of Metropolis algorithms, evaluated at decreasing values of the temperature.

```
procedure SIMULATED_ANNEALING
begin
    $i$ := initial solution
    $c$ := initial value
    repeat
        for $l := 1$ to $L$ do
        begin
            probabilistically generate neighbor $j$ of $i$
            if $f(j) \leq f(i)$ then accept $j$
            else accept $j$ with probability
                $\exp\left(\frac{f(i)-f(j)}{c}\right)$
        end
        update $L$
        decrease $c$
    until stopcriterion
end;
```

FIGURE 25.1 The simulated annealing algorithm in pseudo-code.

We now formulate simulated annealing in terms of a local search algorithm. For an instance (S, f) of a combinatorial optimization problem and a neighborhood function N, Figure 25.1 describes simulated annealing in pseudocode.

The algorithm generates neighbors randomly. If the cost of a neighbor j is at most the cost of the current solution i, then j is always accepted. If neighbor j has higher cost than i, then j is still accepted with a positive probability of

$$\exp\left(\frac{f(i) - f(j)}{c}\right)$$

where c is a control parameter that plays the role of the temperature. The probability of accepting a deterioration in cost depends on the value of the control parameter c: the higher the value of the control parameter, the higher the probability of accepting the deterioration.

The value of the control parameter is decreased during the execution of the algorithm. In Figure 25.1 the value L specifies the number of iterations that the control parameter is kept constant before it is decreased. The values of c and L and the stop criterion are specified by the "cooling schedule."

Initially, at large values of c, large deteriorations will be accepted; as c decreases, only smaller deteriorations will be accepted and, finally, as the value of c approaches 0, no deteriorations will be accepted at all. Note that there is no limitation on the size of deterioration with respect to its acceptance. In simulated annealing, arbitrarily large deteriorations are accepted with positive probability; for these deteriorations the acceptance probability is small, however. This feature means that simulated annealing, in contrast to iterative improvement, can escape from local minima while it still exhibits the favorable features of iterative improvement, namely simplicity and general applicability. The speed of convergence of simulated annealing is determined by the cooling schedule. In Section 25.4, we will indicate that under certain mild conditions on the choice of the cooling schedule simulated annealing converges asymptotically to the set of globally optimal solutions.

Comparing simulated annealing with iterative improvement it is evident that simulated annealing can be viewed as a generalization. Simulated annealing becomes identical to iterative improvement in the case where the value of the control parameter is taken equal to 0. With respect to a comparison between the performances of both algorithms we mention that for most problems simulated annealing performs better than iterative improvement, in the case that we give iterative improvement the same amount of time by repeating it for a number of different initial solutions.

The remainder of this chapter is organized as follows. Section 25.3 is devoted to practical cooling schedules. The asymptotic convergence of simulated annealing is discussed in Section 25.4. As an intermediate result, we mention in Section 25.4 that if the value of the control parameter is kept constant, then the probability distribution of the solutions converges to a unique stationary distribution. In Section 25.5 we derive some characteristic features for simulated annealing in the case that this stationary distribution is attained. We conclude Section 25.6 with some final remarks.

25.3 Practical Implementations

To implement simulated annealing we have to specify the cooling schedule that governs the convergence of the algorithm. A cooling schedule specifies a finite sequence of values of the control parameter and the number of iterations that are executed at each value of the control parameter. More precisely, it is specified by

- an initial value of the control parameter c_0,
- a decrement function for lowering the value of the control parameter,
- a final value of the control parameter specified by a stop criterion, and
- for each value of the control parameter a finite number that specifies the number of iterations that are made with this value of the control parameter.

The search for adequate cooling schedules has been the subject of many studies. Reviews are given by Van Laarhoven and Aarts [4], Collins et al. [5], and Romeo and Sangiovanni-Vincentelli [6]. A more recent empirical study on several cooling schedules is given by Triki et al. [7]. Below we discuss some results.

Most of the existing work on cooling schedules presented in the literature deals with heuristic schedules. We distinguish between two broad classes: static and dynamic schedules. In a static cooling schedule the parameters are fixed; they cannot be changed during execution of the algorithm. In a dynamic cooling schedule the parameters are adaptively changed during execution of the algorithm. Below we illustrate both types of cooling schedules.

25.3.1 Static Cooling Schedules

The following simple schedule is known as the geometric schedule. It originates from the early work on cooling schedules by Kirkpatrick et al. [1], and is still used in many practical situations.

Initial value of the control parameter. To ensure that initially a sufficiently large number of solutions is accepted, one may choose $c_0 = \Delta f_{\max}$, where $\Delta f_{\max}$ is the maximal difference in cost between any two neighboring solutions. Exact calculation of $\Delta f_{\max}$ is quite time-consuming in many cases. However, one often can give simple estimates of its value.

Lowering the control parameter value. The decrement function is given by

$$c_{k+1} = \alpha \cdot c_k, \quad k = 0, 1, \ldots$$

where α is a positive constant smaller than but close to 1. Typical values lie between 0.8 and 0.99.

Final value of the control parameter. The final value is fixed at some small value, which may be related to the smallest possible difference in cost between two neighboring solutions.

Number of iterations that the control parameter is kept constant. This number is fixed and may be related to the size of the neighborhoods in the problem instance at hand.

25.3.2 Dynamic Cooling Schedules

There exist many extensions of the simple static schedule presented above that lead to a dynamic schedule. For instance, a sufficiently large value of c_0 may be obtained by starting off at a small positive value of c_0 and multiplying it with a constant factor, larger than 1, until the fraction $\omega(c_0)$ of accepted solutions is close to 1. Typical values of $\omega(c_0)$ lie between 0.9 and 0.99. An adaptive calculation of the final value of the control parameter may be obtained by terminating the execution of the algorithm if the best solution cost obtained at value c of the control parameter remains unchanged for a number of consecutive values of c. Clearly such a value exists for each local minimum that is found. The number of iterations that the control parameter is kept constant may be determined by requiring that at each value c of the control parameter, a minimum number of solutions is accepted. However, since solutions are accepted with decreasing probability, one would obtain $L \to \infty$ for $c \downarrow 0$. Therefore, L is usually bounded by some constant $L_{\max}$ to avoid that c is kept constant for an extremely large number of iterations.

In addition to this basic dynamic schedule the literature presents a number of more elaborate schedules. Most of these schedules are based on a statistical analysis of simulated annealing using the equilibrium statistics of Section 25.5.

25.4 Asymptotic Convergence

Simulated annealing can be mathematically modeled by means of Markov chains. For details on Markov chain theory we refer to Feller [8], Isaacson and Madsen [9], and Seneta [10]. In this model, we view simulated annealing as a Markov chain that consists of a sequence of trials, where the outcome of the kth trial corresponds to the solution that simulated annealing visits in the kth iteration.

Let (S, f) be a problem instance, N a neighborhood function, $\mathbf{X}(k)$ a random variable denoting the outcome of the kth trial, and c_k the value of the control parameter at the kth trial. Then the transition probability at the kth trial for each pair $i, j \in S$ of outcomes is defined as

$$P_{ij}(k) = \mathbb{P}\{\mathbf{X}(k) = j | \mathbf{X}(k-1) = i\}$$

$$= \begin{cases} G_{ij}(c_k)A_{ij}(c_k) & \text{if } i \neq j \\ 1 - \sum\limits_{l \in S, l \neq i} G_{il}(c_k)A_{il}(c_k) & \text{if } i = j \end{cases} \tag{25.2}$$

where $G_{ij}(c_k)$ denotes the generation probability, that is, the probability of generating a solution j when being at solution i, and $A_{ij}(c_k)$ denotes the acceptance probability, that is, the probability of accepting solution j once it is generated from solution i. The most frequently used choices for these probabilities are the following [11]:

$$G_{ij}(c_k) = \begin{cases} |N(i)|^{-1} & \text{if } j \in N(i) \\ 0 & \text{if } j \notin N(i) \end{cases} \tag{25.3}$$

and

$$A_{ij}(c_k) = \begin{cases} 1 & \text{if } f(j) \leq f(i) \\ \exp((f(i) - f(j))/c_k) & \text{if } f(j) > f(i) \end{cases} \tag{25.4}$$

Instead of one long Markov chain, we can also view simulated annealing as a process in which a sequence of Markov chains is generated, one for each value of the control parameter. In this case, the transition probabilities of each Markov chain are independent of k, in which case the resulting Markov chain is called time-independent or homogeneous. We now focus on this homogeneous model.

Using the theory of Markov chains it is fairly straightforward to show that, under the condition that the neighborhood graph is strongly connected and finite and not all solutions are optimal—in which case the Markov chain is irreducible and aperiodic—there exist a unique stationary distribution of the outcomes to which the Markov chain converges. Under the additional condition that the generation matrix satisfies $G_{ij} = G_{ji}$ for all solutions $i, j \in S$, this distribution assumes the following form [11], where S^* denotes the set of globally optimal solutions.

Theorem 25.1

Let (S, f) be an instance of a combinatorial optimization problem with $S^* \neq S$ and S finite. Furthermore, let N be a neighborhood function that induces a strongly connected neighborhood graph. If the generation matrix *satisfies $G_{ij} = G_{ji}$ for all $i,\ j \in S$, then, after a sufficiently large number of transitions at a fixed value c of the control parameter, applying the transition probabilities of (25.2), (25.3), and (25.4), simulated annealing will find a solution $i \in S$ with a probability given by*

$$\mathbb{P}_c\{\mathbf{X} = i\} \overset{def}{=} q_i(c) = \frac{1}{N_0(c)} \exp\left(-\frac{f(i)}{c}\right) \tag{25.5}$$

where $\mathbf{X}$ is a stochastic variable denoting the current solution obtained by simulated annealing and

$$N_0(c) = \sum_{j \in S} \exp\left(-\frac{f(j)}{c}\right)$$

denotes a normalization constant.

A proof of this theorem is considered beyond the scope of this chapter. For those interested we refer to Aarts and Korst [11]. The probability distribution of (25.5), which besides stationary distribution is also called equilibrium distribution, is the equivalent of the Boltzmann distribution of (25.1). We can now formulate the following important result.

Theorem 25.2

Let the components of distribution $q(c)$ be given by (25.5). Then

$$\lim_{c \downarrow 0} q_i(c) \overset{def}{=} q_i^* = \frac{1}{|S^*|} \chi_{(S^*)}(i)$$

where for any two sets A and $A' \subseteq A$ the characteristic function χ is defined such that $\chi_{(A')}(a) = 1$ if $a \in A'$ and $\chi_{(A')}(a) = 0$ if $a \in A \setminus A'$.

Proof

Using the fact that for all $a \leq 0$, $\lim_{x \downarrow 0} e^{\frac{a}{x}} = 1$ if $a = 0$, and 0 otherwise, we obtain

$$\begin{aligned}
\lim_{c \downarrow 0} q_i(c) &= \lim_{c \downarrow 0} \frac{\exp\left(-\frac{f(i)}{c}\right)}{\sum_{j \in S} \exp\left(-\frac{f(j)}{c}\right)} \\
&= \lim_{c \downarrow 0} \frac{\exp\left(\frac{f^* - f(i)}{c}\right)}{\sum_{j \in S} \exp\left(\frac{f^* - f(j)}{c}\right)} \\
&= \lim_{c \downarrow 0} \frac{1}{\sum_{j \in S} \exp\left(\frac{f^* - f(j)}{c}\right)} \chi_{(S^*)}(i) \\
&\quad + \lim_{c \downarrow 0} \frac{\exp\left(\frac{f^* - f(i)}{c}\right)}{\sum_{j \in S} \exp\left(\frac{f^* - f(j)}{c}\right)} \chi_{(S \setminus S^*)}(i) \\
&= \frac{1}{|S^*|} \chi_{(S^*)}(i) + \frac{0}{|S^*|} \chi_{(S \setminus S^*)}(i),
\end{aligned}$$

which completes the proof. □

The results of Theorems 25.1 and 25.2 are quite interesting since they guarantee asymptotic convergence of the simulated annealing algorithm to the set of globally optimal solutions under the condition that the stationary distribution of (25.5) is attained at each value of c. More specifically, it implies that asymptotically optimal solutions are obtained. This can be expressed as

$$\lim_{c \downarrow 0} \lim_{k \to \infty} \mathbb{P}_c\{\mathbf{X}(k) \in S^*\} = 1$$

We note that it can be proved that this convergence result still holds if we replace the constraint that the neighborhood function satisfies $|N(i)| = |N(j)|$ for all $i,\ j \in S$ by the constraint that the neighborhood graph has to be symmetric [12,29].

Of course, it is not possible to implement simulated annealing such that it consists of a sequence of infinitely long Markov chains. However, we now indicate that simulated annealing also converges in a more realistic inhomogeneous model in which simulated annealing is modeled by a single inhomogeneous Markov chain, where the value of the control parameter may be changed after each iteration. Necessary and sufficient conditions for asymptotic convergence in this case have been derived by Hajek [13]. To discuss this result we need the following definitions.

For any two solutions $i,\ j \in S,\ j$ is reachable at height h from i if there exists a sequence of solutions $i = l_0, l_1, \ldots, l_p = j \in S$ with $G_{l_k,l_{k+1}}(c) > 0$ for $k = 0, \ldots, p-1$ and $f(l_k) \leq h$ for all $k = 0, \ldots, p$.

The depth $d(\hat{\imath})$ of a local optimum $\hat{\imath}$ is the smallest positive number x, such that there is a solution $j \in S$ with $f(j) < f(\hat{\imath})$ that is reachable at height $f(\hat{\imath}) + x$ from $\hat{\imath}$. By definition, for an optimal solution i^*, $d(i^*) = \infty$.

We now can formulate Hajek's result.

Theorem 25.3

Given are an instance $(S,\ f)$ of a combinatorial optimization problem with S finite and a suitable neighborhood function. Furthermore, let $\{c_k\}_{k=1}^{\infty}$ be a sequence of values of the control parameter satisfying

$$c_k = \frac{\Gamma}{\log(k+1)}, \quad k = 0, 1, \ldots$$

for some constant Γ. Then asymptotic convergence of simulated annealing to the set of globally optimal solutions, using the transition probabilities of (25.2), (25.3), and (25.4), is guaranteed if and only if

- *the neighborhood graph is strongly connected,*
- *i is reachable from j at height h if and only if j is reachable from i at height h, for arbitrary $i,\ j \in S$ and h, and*
- *the constant Γ satisfies $\Gamma \geq D$, where D is the depth of the deepest local, nonglobal minimum.*

Kern [14] has addressed the problem of calculating the value of D. In particular, he showed for a number of problems that it is unlikely that D can be calculated in polynomial time for arbitrary instances. Kern also presents bounds on the value of D for several combinatorial optimization problems.

Several authors have addressed the convergence of simulated annealing for more general forms of the generation and acceptance probabilities than the probabilities of (25.3) and (25.4). Especially the use of more general forms of the acceptance probability has been studied extensively with the aim to find probabilities different from the exponential form of the Metropolis criterion that exhibit a similar asymptotic convergence behavior. However, this has not resulted in practical acceptance probabilities that are essentially different from the ones used above.

Summarizing, simulated annealing can find optimal solutions with probability 1 if it is allowed an infinite number of transitions. This result is mainly of theoretical interest. The real strength of simulated annealing lies in the good practical results that can be obtained by applying more efficient finite-time implementations, which we discussed in Section 25.3. Several papers derive theoretical results for finite-time implementations of simulated annealing [15–27]. These results are mainly concerned with the application of simulated annealing to a particular combinatorial optimization problem and not with the algorithm as a whole as is the case for the convergence result discussed.

25.5 Equilibrium Statistics

To enhance our understanding of the simulated annealing algorithm, we discuss in this section some characteristic features of the algorithm in the case that we are at equilibrium. We will assume that all conditions of Theorem 25.1 are satisfied. This means that the equilibrium is given by the stationary distribution defined by (25.5), that is, at any value c of the control parameter, the probability of being in solution $i \in S$ is given by $q_i(c) = \frac{1}{N_0(c)} \exp(-\frac{f(i)}{c})$, where $N_0(c) = \sum_{j \in S} \exp(-\frac{f(j)}{c})$. We can write the expected cost $\langle f \rangle_c$ and the variance σ_c^2 of the cost as

$$\langle f \rangle_c = \sum_{i \in S} f(i) q_i(c) \tag{25.6}$$

and

$$\sigma_c^2 = \sum_{i \in S} (f(i) - \langle f \rangle_c)^2 \, q_i(c) \tag{25.7}$$

respectively. Note that because $S \neq S^*$ by one of the conditions of Theorem 25.1 and because $q_i(c) > 0$ for all $c > 0$, we have $\sigma_c^2 > 0$. Using this observation and the following theorem yields that if the control parameter decreases, then the expected cost at equilibrium also decreases.

Theorem 25.4

Let $\langle f \rangle_c$ and σ_c^2 be defined by (25.6) and (25.7), respectively. Then, we have

$$\frac{\partial}{\partial c} \langle f \rangle_c = \frac{\sigma_c^2}{c^2}$$

Proof

Using the definition of $\langle f \rangle_c$ gives

$$\frac{\partial}{\partial c} \langle f \rangle_c = \sum_{i \in S} f(i) \frac{\partial}{\partial c} q_i(c) \tag{25.8}$$

Furthermore, simple calculus yields

$$\frac{\partial}{\partial c} N_0(c) = \sum_{j \in S} \frac{f(j)}{c^2} \exp\left(\frac{-f(j)}{c}\right)$$

and thus, by using $\frac{\partial}{\partial c}(g(c) \cdot h(c)) = g(c)\frac{\partial}{\partial c}h(c) + h(c)\frac{\partial}{\partial c}g(c)$ with $g(c) = 1/N_0(c)$ and $h(c) = \exp\left(\frac{-f(i)}{c}\right)$,

$$\begin{aligned}
\frac{\partial}{\partial c} q_i(c) &= \frac{\partial}{\partial c} \frac{\exp\left(\frac{-f(i)}{c}\right)}{N_0(c)} \\
&= \frac{f(i)}{c^2} \frac{\exp\left(\frac{-f(i)}{c}\right)}{N_0(c)} - \frac{\exp\left(\frac{-f(i)}{c}\right)}{N_0^2(c)} \frac{\partial}{\partial c} N_0(c) \\
&= \frac{q_i(c)}{c^2} f(i) - \frac{q_i(c)}{c^2} \frac{\sum_{j \in S} f(j) \exp\left(\frac{-f(j)}{c}\right)}{N_0(c)} \\
&= \frac{q_i(c)}{c^2} (f(i) - \langle f \rangle_c)
\end{aligned}$$

Substituting this for $\frac{\partial}{\partial c} q_i(c)$ in (25.8) now gives

$$\begin{aligned}
\frac{\partial}{\partial c} \langle f \rangle_c &= \frac{1}{c^2} \sum_{i \in S} q_i(c) \left(f^2(i) - f(i) \langle f \rangle_c \right) \\
&= \frac{1}{c^2} \left(-\langle f \rangle_c^2 + \sum_{i \in S} f^2(i) q_i(c) \right)
\end{aligned}$$

By definition, we have $\langle f\rangle_c = \sum_{i\in S} f(i)q_i(c)$. Furthermore, because $\sum_{i\in S} q_i(c) = 1$ we also have $\langle f\rangle_c = \sum_{i\in S}\langle f\rangle_c q_i(c)$. Using these observations we obtain

$$\begin{aligned}\frac{\partial}{\partial c}\langle f\rangle_c &= \frac{1}{c^2}\left(-2\langle f\rangle_c^2 + \langle f\rangle_c^2 + \sum_{i\in S} f^2(i)q_i(c)\right)\\ &= \frac{1}{c^2}\sum_{i\in S}\left(f^2(i) - 2f(i)\langle f\rangle_c + \langle f\rangle_c^2\right)q_i(c)\\ &= \frac{1}{c^2}\sum_{i\in S}(f(i) - \langle f\rangle_c)^2 q_i(c)\\ &= \frac{\sigma_c^2}{c^2}\end{aligned}$$

□

By using the following theorem, we can strengthen the observation that the expected cost at equilibrium decreases if the control parameter decreases. It implies that the expected cost at equilibrium decreases from some value that is at most the average cost to the optimal cost.

Theorem 25.5

Let $\langle f\rangle_c$ and σ_c^2 be defined by (25.6) and (25.7), respectively. Furthermore, let $\langle f\rangle_\infty = \frac{1}{|S|}\sum_{i\in S} f(i)$ denote the average cost and let f^* denote the optimal cost. Then, we have

$$\begin{aligned}\lim_{c\to\infty}\langle f\rangle_c &= \langle f\rangle_\infty\\ \lim_{c\downarrow 0}\langle f\rangle_c &= f^*\\ \lim_{c\to\infty}\sigma_c^2 &= \frac{1}{|S|}\sum_{i\in S}(f(i) - \langle f\rangle_\infty)^2\\ \lim_{c\downarrow 0}\sigma_c^2 &= 0\end{aligned}$$

Proof

The theorem can be proved by using the definitions of $\langle f\rangle_c$, σ_c^2, and $q(c)$. □

In the above two theorems, we studied the solutions space as a whole. For an individual solution $i \in S$, we can derive the following result. For decreasing c, the probability of being in solution i at equilibrium increases if i is optimal and it decreases if i has a cost that is at least the average cost. Otherwise, a turning point c_i exists, such that the probability of being in solution i increases if $c > c_i$ and it decreases if $c < c_i$.

Theorem 25.6

Let $\langle f\rangle_\infty = \frac{1}{|S|}\sum_{i\in S} f(i)$ denote the average cost, and let f^* denote the optimal cost. Furthermore, let the components of distribution $q(c)$ of S be given by (25.5). If $S \neq S^*$, then we have for any $i \in S$

$$\frac{\partial}{\partial c}q_i(c) < 0$$

if $f(i) = f^*$,

$$\frac{\partial}{\partial c}q_i(c) > 0$$

if $f(i) \geq \langle f\rangle_\infty$, and

$$\frac{\partial}{\partial c}q_i(c)\begin{cases} < 0 & \text{if } c > c_i\\ = 0 & \text{if } c = c_i\\ > 0 & \text{if } c < c_i\end{cases}$$

for some $c_i > 0$ if $f^* < f(i) < \langle f\rangle_\infty$.

Proof
In Theorem 25.4 we proved

$$\frac{\partial}{\partial c} q_i(c) = \frac{q_i(c)}{c^2}(f(i) - \langle f \rangle_c)$$

As $\frac{q_i(c)}{c^2} > 0$, this implies that the sign of $\frac{\partial}{\partial c} q_i(c)$ is determined by the sign of $f(i) - \langle f \rangle_c$. As we concluded from Theorems 25.4 and 25.5, $\langle f \rangle_c$ increases from f^* to $\langle f \rangle_\infty$ when c increases. The theorem now follows easily. □

25.6 Conclusion

Simulated annealing is a local search algorithm that uses randomization to escape from local optima. Since its introduction in 1983 simulated annealing has been applied to a large amount of different problems in many different areas. More than 20 years of experience had led to the following general observations:

- High-quality solutions can be obtained but sometimes at the cost of large amounts of computation time.
- In many practical situations, where no tailored algorithms are available, simulated annealing is a valuable algorithm due to its general applicability and its ease of implementation.

References

[1] Kirkpatrick, S., Gelatt Jr., C. D., and Vecchi, M. P., Optimization by simulated annealing, *Science,* 220, 671, 1983.

[2] Černý, V., Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, *J. Optimization Theory Appl.,* 45, 41, 1985.

[3] Binder, K., *Monte Carlo Methods in Statistical Physics,* Springer, Berlin, 1978.

[4] Laarhoven, P. J. M. van and Aarts, E. H. L., *Simulated Annealing: Theory and Applications,* Reidel, Dordrecht, 1987.

[5] Collins, N. E., Eglese, R. W., and Golden, B. L., Simulated annealing: an annotated bibliography, *Am. J. Math. Manage. Sci.,* 8, 209, 1988.

[6] Romeo, F. and Sangiovanni-Vincentelli, A., A theoretical framework for simulated annealing, *Algorithmica,* 6, 302, 1991.

[7] Triki, E., Collette, Y., Siarry, P., A theoretical study on the behavior of simulated annealing leading to a new cooling schedule, *Eur. J. Oper. Res.,* 166, 77, 2005.

[8] Feller, W., *An Introduction to Probability Theory and Its Applications*, Vol. 1, Wiley, New York, 1950.

[9] Isaacson, D. and Madsen, R., *Markov Chains,* Wiley, New York, 1976.

[10] Seneta, E., *Non-negative matrices and Markov chains,* 2nd ed., Springer, Berlin, 1981.

[11] Aarts, E. H. L. and Korst, J. H. M., *Simulated Annealing and Boltzmann Machines,* Wiley, New York, 1989.

[12] Faigle, U. and Kern, W., Note on the convergence of simulated annealing algorithms, *SIAM J. Control Optim.,* 29, 153, 1991.

[13] Hajek, B., Cooling schedules for optimal annealing, *Math. Oper. Res.,* 13, 311, 1988.

[14] Kern, W., On the depth of combinatorial optimization problems, *Discrete Appl. Math.,* 43(2), 115, 1993.

[15] Albrecht, A. A., A problem-specific convergence bound for simulated annealing-based local search, *Proc. Int. Conf. on Computational Science and its Applications,* 2004, p. 405.

[16] Drost, S., Jansen, T., and Wegener, I., Dynamic parameter control in simple evolutionary algorithms, in *Proc. 6th Workshop on Foundations of Genetic Algorithms,* 2001, p. 275.

[17] Jerrum, M., Large cliques elude the Metropolis process, *Random Structures Algorithms,* 3, 347, 1992.

[18] Jerrum, M. and Sorkin, G. B., The Metropolis algorithm for graph bisection, *Discrete Appl. Math.,* 82, 155, 1998.

[19] Mitra, D., Romeo, F., and Sangiovanni-Vincentelli, A. L., Convergence and finite-time behavior of simulated annealing, *Adv. Appl. Probab.,* 18, 747, 1986.

[20] Nolte, A. and Schrader, R., Simulated annealing and its problems to color graphs, *Proc. 4th Annual European Symp. on Algorithms,* 1996, p. 138.

[21] Nolte, A. and Schrader, R., Coloring in sublinear time, *Proc. 5th Annual European Symp. on Algorithms,* 1997, p. 388.

[22] Nolte, A. and Schrader, R., A note on the finite time behavior of simulated annealing, *Math. Oper. Res.,* 25, 476, 2000.

[23] Orosz, J. E. and Jacobson, S. H., Finite-time performance analysis of static simulated annealing algorithms, *Comput. Optim. Appl.,* 21, 21, 2002.

[24] Sasaki, G. and Hajek, B., The time complexity of maximum matching by simulated annealing, *JACM,* 35, 387, 1988.

[25] Sorkin, G. B., Efficient simulated annealing on fractal energy landscapes, *Algorithmica,* 6, 367, 1991.

[26] Steinhöfel, K., Albrecht, A., and Wong, C. K., On various cooling schedules for simulated annealing applied to the job shop problem, *Proc. 2nd Int. Workshop on Randomized and Approximation Techniques in Computer Science,* 1998, p. 260.

[27] Wegener, I., Simulated annealing beats Metropolis in combinatorial optimization, *Proc. 32nd ICALP,* 2005, p. 589.

[28] Metropolis, M., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. Equation of state calculations by fast computing machines. *Journal of Chemical Physics,* 21, 1953, 1087–1092.

[29] Michiels, W., Aarts, E., and Korst, *J. Theoretical Aspects of Local Search*, Springer, 2007.

26

Ant Colony Optimization

Marco Dorigo
Free University of Brussels

Krzysztof Socha
Free University of Brussels

26.1 Introduction

This chapter presents an overview of ant colony optimization (ACO)—a metaheuristic inspired by the behavior of real ants. Ant colony optimization was proposed by Dorigo and colleagues [1–3] as a method for solving hard combinatorial optimization problems (COPs).

Ant colony optimization algorithms are part of *swarm intelligence*, that is, the research field that studies algorithms inspired by the observation of the behavior of *swarms*. Swarm intelligence algorithms are made up of simple individuals that cooperate through self-organization, that is, without any form of central control over the swarm members. A detailed overview of the self-organization principles exploited by these algorithms, as well as examples from biology, can be found in Ref. [4]. Many swarm intelligence algorithms have been proposed in the literature. For an overview of the field of swarm intelligence, we refer the interested reader to Ref. [5].

This chapter, which is dedicated to present a concise overview of ACO, is organized as follows. Section 26.2 presents the biological phenomenon that provided the original inspiration. Section 26.3 presents a formal description of the ACO metaheuristic. Section 26.4 overviews the most popular variants of ACO and gives examples of their application. Section 26.5 shows current research directions, and Section 26.6 summarizes and concludes the chapter.

26.2 From Biology to Algorithms

Ant colony optimization was inspired by the observation of the behavior of real ants. In this section, we present a number of observations made in experiments with real ants, and then we show how these observations inspired the design of the ACO metaheuristic.

26.2.1 Ants

One of the first researchers to investigate the social behavior of insects was the French entomologist Pierre-Paul Grassé. In the 1940s and 1950s, he was observing the behavior of termites—in particular, the *Bellicositermes natalensis* and *Cubitermes* species. He discovered [6] that these insects are capable to react to what he called "significant stimuli," signals that activate a genetically encoded reaction. He observed [7] that the effects of these reactions can act as new significant stimuli for both the insect that produced them and for the other insects in the colony. Grassé used the term *stigmergy* [7] to describe this particular type of indirect communication in which "the workers are stimulated by the performance they have achieved."

The two main characteristics of stigmergy that differentiate it from other means of communication are:

- the physical, nonsymbolic nature of the information released by the communicating insects, which corresponds to a modification of physical environmental states visited by the insects; and
- the local nature of the released information, which can only be accessed by those insects that visit the place where it was released (or its immediate neighborhood).

Examples of stigmergy can be observed in colonies of ants. In many ant species, ants walking to—and from—a food source, deposit on the ground a substance called *pheromone*. Other ants are able to smell this pheromone, and its presence influences the choice of their path, that is, they tend to follow strong pheromone concentrations. The pheromone deposited on the ground forms a *pheromone trail*, which allows the ants to find good sources of food that have been previously identified by other ants.

Some researchers investigated experimentally this pheromone laying and following behavior to better understand it and to be able to quantify it. Deneubourg et al. [8] set up an experiment called a "binary bridge experiment." They used *Linepithema humile* ants (also known as Argentine ants). The ants' nest was connected to a food source by two bridges of equal length. The ants could freely choose which bridge to use when searching for food and bringing it back to the nest. Their behavior was then observed over a period of time.

In this experiment, initially there is no pheromone on the two bridges. The ants start exploring the surroundings of the nest and eventually cross one of the bridges and reach the food source. When walking to the food source and back, the ants deposit pheromone on the bridge they use. Initially, each ant randomly chooses one of the bridges. However, because of random fluctuations, after some time there will be more pheromone deposited on one of the bridges than on the other. Because ants tend to prefer in probability to follow a stronger pheromone trail, the bridge that has more pheromone will attract more ants. This in turn makes the pheromone trail grow stronger, until the colony of ants converges toward the use of a same bridge.[1]

This colony level behavior, based on autocatalysis, that is, on the exploitation of positive feedback, can be exploited by ants to find the shortest path between a food source and their nest. This was demonstrated in another experiment conducted by Goss et al. [9], in which the two bridges were not of the same length: one was significantly longer than the other. In this case, the stochastic fluctuations in the initial choice of a bridge were much reduced as a second mechanism played an important role: those ants choosing by chance the shorter bridge were also the first to reach the nest, and when returning to the nest, they chose the shorter bridge with higher probability as it had a stronger pheromone trail. Therefore, the ants—thanks to the pheromone following and depositing mechanism—quickly converged to the use of the shorter bridge.

In the next section we explain how these experiments and findings were used to develop optimization algorithms.

[1] Deneubourg et al. [8] conducted several experiments, and results show that each of the two bridges was used in about 50% of the cases.

26.2.2 Algorithms

Stimulated by the interesting results of the experiments described in the previous section, Goss et al. [9] developed a model to explain the behavior observed in the binary bridge experiment. Assuming that after t time units since the start of the experiment, m_1 ants had used the first bridge and m_2 the second one, the probability p_1 for the $(m+1)$th ant to choose the first bridge can be given by

$$p_{1(m+1)} = \frac{(m_1+k)^h}{(m_1+k)^h + (m_2+k)^h} \tag{26.1}$$

where parameters k and h are needed to fit the model to the experimental data. The probability that the same $(m+1)$th ant chooses the second bridge is $p_{2(m+1)} = 1 - p_{1(m+1)}$. Monte Carlo simulations, run to test whether the model corresponds to the real data [10], showed very good fit for $k \approx 20$ and $h \approx 2$.

This basic model, which explains the behavior of real ants, may be used as an inspiration to design artificial ants that solve optimization problems defined in a similar way. In the above described *ant foraging behavior* example, stigmergic communication happens via the pheromone that ants deposit on the ground. Analogously, artificial ants may simulate pheromone laying by modifying appropriate pheromone variables associated with problem states they visit while building solutions to the optimization problem. Also, according to the stigmergic communication model, the artificial ants would have only local access to these pheromone variables.

Therefore, the main characteristics of stigmergy mentioned in the previous section can be extended to artificial agents by

- associating state variables with different problem states; and
- giving the agents only local access to these variables.

Another important aspect of real ants' foraging behavior that may be exploited by artificial ants is the coupling between the autocatalytic mechanism and the *implicit evaluation* of solutions. By implicit solution evaluation, we mean the fact that shorter paths (which correspond to lower cost solutions in the case of artificial ants) are completed earlier than longer ones, and therefore they receive pheromone reinforcement quicker. Implicit solution evaluation coupled with autocatalysis can be very effective: the shorter the path, the sooner the pheromone is deposited, and the more ants use the shorter path. If appropriately used, it can be a powerful mechanism in population-based optimization algorithms (e.g., in evolutionary algorithms [11,12] autocatalysis is implemented by the selection/reproduction mechanism).

Stigmergy, together with implicit solution evaluation and autocatalytic behavior, gave rise to ACO. The basic idea of ACO follows very closely the biological inspiration. Therefore, there are many similarities between real and artificial ants. Both real and artificial ant colonies are composed of a population of individuals that work together to achieve a certain goal. A colony is a population of simple, independent, asynchronous agents that cooperate to find a good *solution* to the problem at hand. In the case of real ants, the problem is to find the food, while in the case of artificial ants, it is to find a good solution to a given optimization problem. A single ant (either a real or an artificial one) is able to find a solution to its problem, but only cooperation among many individuals through stigmergy enables them to find *good* solutions.

In the case of real ants, they deposit and react to a chemical substance called *pheromone*. Real ants simply deposit it on the ground while walking. Artificial ants live in a *virtual* world, hence they only modify numeric values (called for analogy *artificial pheromones*) associated with different problem states. A sequence of pheromone values associated with problem states is called *artificial pheromone trail*. In ACO, the artificial pheromone trails are the sole means of communication among the ants. A mechanism analogous to the evaporation of the physical pheromone in real ant colonies allows the artificial ants to *forget* the history and focus on new promising search directions.

Just like real ants, artificial ants create their solutions sequentially by moving from one problem state to another. Real ants simply walk, choosing a direction based on local pheromone concentrations and a stochastic decision policy. Artificial ants also create solutions step by step, moving through available problem states and making stochastic decisions at each step.

There are however some important differences between real and artificial ants:

- Artificial ants live in a discrete world—they move sequentially through a finite set of problem states.
- The pheromone update (i.e., pheromone depositing and evaporation) is not accomplished in exactly the same way by artificial ants as by real ones. Sometimes the pheromone update is done only by some of the artificial ants, and often *only after* a solution has been constructed.
- Some implementations of artificial ants use additional mechanisms that do not exist in the case of real ants. Examples include look-ahead, local search, backtracking, etc.

26.3 The Ant Colony Optimization Metaheuristic

Ant colony optimization has been formalized into a combinatorial optimization metaheuristic by Dorigo et al. [13,14] and has since been used to tackle many combinatorial optimization problems (COPS).

Given a COP, the first step for the application of ACO to its solution consists in defining an adequate model. This is then used to define the central component of ACO: the pheromone model. The model of a COP may be defined as follows:

Definition 26.1

A model $P = (\mathbf{S}, \mathbf{\Omega}, f)$ *of a COP consists of*

- *a search space* $\mathbf{S}$ *defined over a finite set of discrete decision variables and a set* $\mathbf{\Omega}$ *of constraints among the variables;*
- *an objective function* $f : \mathbf{S} \to \mathbb{R}_0^+$ *to be minimized.*[2]

The search space $\mathbf{S}$ *is defined as follows: Given is a set of discrete variables* X_i, $i = 1, \dots, n$, *with values* $v_i^j \in \mathbf{D}_i = \{v_i^1, \dots, v_i^{|\mathbf{D}_i|}\}$. *A variable instantiation, that is, the assignment of a value* v_i^j *to a variable* X_i, *is denoted by* $X_i \leftarrow v_i^j$. *A solution* $s \in \mathbf{S}$, *that is, a complete assignment in which each decision variable has a value assigned that satisfies all the constraints in the set* $\mathbf{\Omega}$, *is a feasible solution of the given COP. If the set* $\mathbf{\Omega}$ *is empty,* P *is called an unconstrained problem model, otherwise it is said to be constrained. A solution* $s^* \in \mathbf{S}$ *is called a global optimum if and only if* $f(s^*) \leq f(s)\ \forall s \in \mathbf{S}$. *The set of all globally optimal solutions is denoted by* $\mathbf{S}^* \subseteq \mathbf{S}$. *Solving a COP requires finding at least one* $s^* \in \mathbf{S}^*$.

The model of a COP is used to derive the pheromone model used by ACO. First, an instantiated decision variable $X_i = v_i^j$ (i.e., a variable X_i with a value v_i^j assigned from its domain $\mathbf{D}_i$) is called a *solution component* and denoted by c_{ij}. The set of all possible solution components is denoted by $\mathbf{C}$. A pheromone trail parameter T_{ij} is then associated with each component c_{ij}. The set of all pheromone trail parameters is denoted by $\mathbf{T}$. The value of a pheromone trail parameter T_{ij} is denoted by τ_{ij} (and called pheromone value).[3] This pheromone value is then used and updated by the ACO algorithm during the search. It allows modeling the probability distribution of different components of the solution.

In ACO, artificial ants build a solution to a COP by traversing the so-called *construction graph*, $G_C(\mathbf{V}, \mathbf{E})$. The fully connected construction graph consists of a set of vertices $\mathbf{V}$ and a set of edges $\mathbf{E}$. The set of components $\mathbf{C}$ may be associated either with the set of vertices $\mathbf{V}$ of the graph G_C, or with the set of its edges $\mathbf{E}$. The ants move from vertex to vertex along the edges of the graph, incrementally building a *partial solution.* Additionally, the ants deposit a certain amount of pheromone on the components, that is, either on the vertices or on the edges that they traverse. The amount $\Delta\tau$ of pheromone deposited may depend

[2] Note that minimizing over an objective function f is the same as maximizing over $-f$. Therefore, every COP can be described as a minimization problem.

[3] Note that pheromone values are in general a function of the algorithm's iteration $t : \tau_{ij} = \tau_{ij}(t)$.

on the quality of the solution found. Subsequent ants utilize the pheromone information as a guide toward more promising regions of the search space.

The ACO metaheuristic is shown in Algorithm 26.1. It consists of an initialization step and a loop over three algorithmic components. A single iteration of the loop consists of constructing solutions by all ants, their (optional) improvement with the use of a local search algorithm, and an update of the pheromones. In the following, we explain these three algorithmic components in more detail.

Algorithm 26.1 Ant colony optimization metaheuristic

```
Set parameters, initialize pheromone trails
while termination conditions not met do
  ConstructAntSolutions
  ApplyLocalSearch        {optional}
  UpdatePheromones
end while
```

ConstructAntSolutions

A set of m artificial ants construct solutions from elements of a finite set of available solution components $\mathbf{C} = \{c_{ij}\}$, $i = 1, \ldots, n$, $j = 1, \ldots, |\mathbf{D}_i|$. A solution construction starts with an empty partial solution $s^p = \emptyset$. Then, at each construction step, the current partial solution s^p is extended by adding a feasible solution component from the set of feasible neighbors $\mathbf{N}(s^p) \subseteq \mathbf{C}$. The process of constructing solutions can be regarded as a path on the construction graph $G_C = (\mathbf{V}, \mathbf{E})$. The allowed paths in G_C are implicitly defined by the solution construction mechanism that defines the set $\mathbf{N}(s^p)$ with respect to a partial solution s^p.

The choice of a solution component from $\mathbf{N}(s^p)$ is done probabilistically at each construction step. The exact rules for the probabilistic choice of solution components vary across different ACO variants. The best known rule is the one of ant system (AS) [3]:

$$p(c_{ij}|s^p) = \frac{\tau_{ij}^{\alpha} \cdot \eta(c_{ij})^{\beta}}{\sum_{c_{il} \in \mathbf{N}(s^p)} \tau_{il}^{\alpha} \cdot \eta(c_{il})^{\beta}}, \quad \forall c_{ij} \in \mathbf{N}(s^p) \tag{26.2}$$

where τ_{ij} is the pheromone value associated with the component c_{ij}, and $\eta(\cdot)$ is a function that assigns at each construction step a heuristic value to each feasible solution component $c_{ij} \in \mathbf{N}(s^p)$. The values that are returned by this function are commonly called *heuristic information.* Furthermore, α and β are positive parameters, whose values determine the relative importance of pheromone versus heuristic information. Eq. (26.2) is a generalization of Eq. (26.1) presented in Section 26.2: ACO formalization follows closely the biological inspiration.

ApplyLocalSearch

Once solutions have been constructed, and before updating pheromones, often some optional actions may be required. These are often called *daemon actions*, and can be used to implement problem specific and/or centralized actions, which cannot be performed by single ants. The most used daemon action consists in the application of local search to the constructed solutions: the locally optimized solutions are then used to decide which pheromones to update.

UpdatePheromones

The aim of the pheromone update is to increase the pheromone values associated with good or promising solutions, and to decrease those that are associated with bad ones. Usually, this is achieved (i) by decreasing all the pheromone values through *pheromone evaporation*, and (ii) by increasing the pheromone levels associated with a chosen set of good solutions $\mathbf{S}_{upd}$:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho \sum_{s \in \mathbf{S}_{upd} | c_{ij} \in s} F(s) \tag{26.3}$$

where $\mathbf{S}_{upd}$ is the set of solutions that are used for the update, $\rho \in (0, 1]$ is a parameter called evaporation rate, and $F : \mathbf{S} \to \mathbb{R}_0^+$ a function such that $f(s) < f(s') \Rightarrow F(s) \geq F(s'), \forall s \neq s' \in \mathbf{S}$. $F(\cdot)$ is commonly called the *fitness function*.

Pheromone evaporation is needed to avoid a too rapid convergence of the algorithm. It implements a useful form of *forgetting*, favoring the exploration of new areas in the search space. Different ACO algorithms, for example, Ant Colony System (ACS) [15] or $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [16] differ in the way they update the pheromone.

Instantiations of the update rule presented in Eq. (26.3) are obtained by different specifications of $\mathbf{S}_{upd}$, which in many cases is a subset of $\mathbf{S}_{iter} \cup \{s_{bs}\}$, where $\mathbf{S}_{iter}$ is the set of solutions that were constructed in the current iteration, and s_{bs} the *best-so-far* solution, that is, the best solution found since the first algorithm iteration. A well-known example is the AS-update rule, that is, the update rule of AS [3], where

$$\mathbf{S}_{upd} \leftarrow \mathbf{S}_{iter} \tag{26.4}$$

An example of a pheromone update rule that is more often used in practice is the IB-update rule (where IB stands for *iteration best*):

$$\mathbf{S}_{upd} \leftarrow \arg\max_{s \in \mathbf{S}_{iter}} F(s) \tag{26.5}$$

The IB-update rule introduces a much stronger bias toward the good solutions found than the AS-update rule. Although this increases the speed with which good solutions are found, it also increases the probability of premature convergence. An even stronger bias is introduced by the BS-update rule, where BS refers to the use of the best-so-far solution s_{bs}. In this case, $\mathbf{S}_{upd}$ is set to $\{s_{sb}\}$. In practice, ACO algorithms that use variants of the IB or the BS-update rules and that additionally include mechanisms to avoid premature convergence achieve better results than those that use the AS-update rule.

26.3.1 Example: The Traveling Salesman Problem

One of the most popular ways to illustrate how the ACO metaheuristic works, is via its application to the traveling salesman problem (TSP). The TSP consists of a set of locations (cities) and a traveling salesman that has to visit all the locations once and only once. The distances between the locations are given and the task is to find a Hamiltonian tour of minimal length. The problem has been proven to be NP-hard [17].

The application of ACO to the TSP is straightforward. The moves between the locations become the solution components, that is, the move from city i to city j becomes a solution component $c_{ij} \equiv c_{ji}$. The construction graph $G_C = (\mathbf{V}, \mathbf{E})$ is defined by associating the set of locations with the set $\mathbf{V}$ of vertices of the graph. Since, in principle, it is possible to move from any city to any other one, the construction graph is fully connected and the number of vertices is equal to the number of locations defined by the problem instance. Furthermore, the lengths of the edges between the vertices are proportional to the distances between the locations represented by these vertices. The pheromone is associated with the set $\mathbf{E}$ of edges of the graph. An example of the resulting construction graph G_C is presented in Figure 26.1(a).

The ants construct the solutions as follows. Each ant starts from a randomly selected location (vertex of the graph G_C). Then, at each construction step it moves along the edges of the graph. Each ant keeps a memory of its path through the graph, and in subsequent steps it chooses among the edges that do not lead to vertices that it has already visited. An ant has constructed a solution once it has visited all the vertices of the graph. At each construction step an ant chooses probabilistically the edge to follow among the available ones (those that lead to yet unvisited vertices). The exact rule depends on the implementation, an example being Eq. (26.2). Once all the ants have completed their tour, the pheromone on the edges is updated according to one of the possible implementations of Eq. (26.3). Ant colony optimization has been shown to perform quite well on the TSP [18].

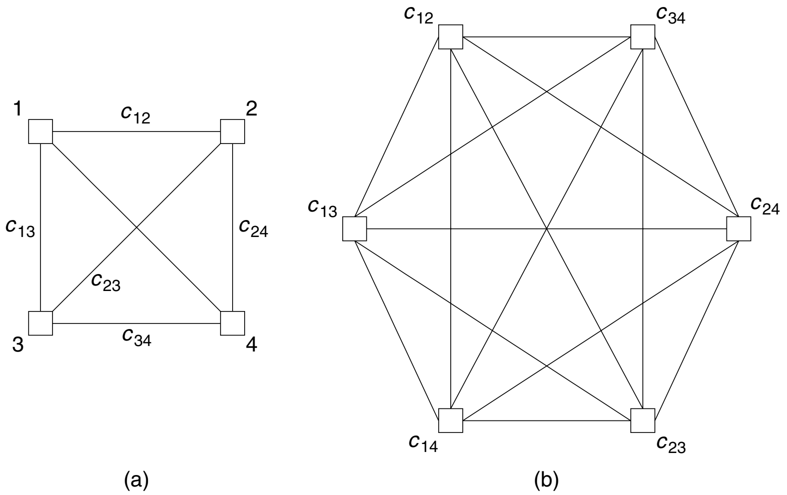


FIGURE 26.1 Example construction graphs for a four-city TSP when (a) components are associated with the edges of the graph, and when (b) components are associated with the vertices of the graph. Note that $c_{ij} \equiv c_{ji}$.

It is worth noticing that it is also possible to associate the set of solution components of the TSP (or any other COP) with the set of vertices **V** rather than the set of edges **E** of the construction graph G_C. For the TSP, this would mean associating the moves between locations with the set **V** of vertices of the construction graph, and the locations with the set **E** of its edges. The corresponding example construction graph for a four-city TSP is presented in Figure 26.1(b). When using this approach, the ants' solution construction process has to be also properly modified: the ants would have to move from vertex to vertex of the construction graph choosing thereby the *connections between the cities*.

It is important to note that both ways of defining the construction graph are correct and both may be used in practice. Although for the TSP the first way seems more intuitive and was in fact used in all the applications of ACO to the TSP that we are aware of, in other cases the second way might be better suited. For example, it was the selected choice in the case of the university course timetabling problem (UCTP) [19].

26.4 Main Variants of ACO

Several variants of ACO have been proposed in the literature. Here we present the three most successful ones, Ant System (AS)—the first implementation of an ACO algorithm—followed by $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) and Ant Colony System (ACS), together with a short list of their applications.

To illustrate the differences between them clearly, we use the example of the TSP, as described in Section 26.3.1.

26.4.1 Ant System

Ant System was the first ACO algorithm to be proposed in the literature [1–3]. Its main characteristic is that the pheromone values are updated by *all* the ants that have completed the tour. The pheromone update for τ_{ij}, that is, for edge joining cities i and j, is performed as follows:

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{26.6}$$

where ρ is the evaporation rate, m is the number of ants, and $\Delta\tau_{ij}^{k}$ is the quantity of pheromone per unit length laid on edge $(i,\ j)$ by the kth ant:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if ant } k \text{ used edge } (i,\ j) \text{ in its tour} \\ 0 & \text{otherwise} \end{cases} \tag{26.7}$$

where Q is a constant and L_k is the tour length of the kth ant.

When constructing the solutions, the ants in AS traverse a construction graph and make a probabilistic decision at each vertex. The transition probability p_{ij}^{k} of the kth ant moving from city i to city j is given by

$$p_{ij}^{k} = \begin{cases} \frac{\tau_{ij}^{\alpha}\cdot\eta_{ij}^{\beta}}{\sum_{l\in \text{ allowed}_k}\tau_{il}^{\alpha}\cdot\eta_{il}^{\beta}} & \text{if } j \in \text{allowed}_k \\ 0 & \text{otherwise} \end{cases} \tag{26.8}$$

where allowed$_k$ is the list of cities not yet visited by the kth ant, and α and β are the parameters that control the relative importance of the pheromone versus the heuristic information η_{ij} given by

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{26.9}$$

where d_{ij} is the length of edge $(i,\ j)$.

Several implementations of the AS algorithm have been applied to different COPs. The first and best known is the application to the TSP [1–3]. However, AS was also used successfully to tackle other combinatorial problems. The AS–QAP [20,21] algorithm was used for solving the quadratic assignment problem (QAP), AS–JSP [22] for the job-shop scheduling problem (JSP), AS–VRP [23,24] for the vehicle routing problem (VRP), and AS–SCS [25,26] for the shortest common supersequence (SCS) problem.

26.4.2 $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System

$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System is an improvement over the original AS idea. $\mathcal{MM}$AS was proposed by Stützle and Hoos [16], who introduced a number of changes of which the most important are the following:

- only the best ant can update the pheromone trails, and
- the minimum and maximum values of the pheromone are limited.

Eq. (26.6) takes the following new form:

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{\text{best}} \tag{26.10}$$

where $\Delta\tau_{ij}^{\text{best}}$ is the pheromone update value defined by

$$\Delta\tau_{ij}^{\text{best}} = \begin{cases} \frac{1}{L_{\text{best}}} & \text{if the best ant used edge } (i,\ j) \text{ in its tour} \\ 0 & \text{otherwise} \end{cases} \tag{26.11}$$

L_{best} is the length of the tour of the best ant. This may be (subject to the algorithm designer decision) either the best tour found in the current iteration—*iteration best*, L_{ib}—or the best solution found since the start of the algorithm—*best-so-far*, L_{bs}—or a combination of both.

Concerning the limits on the minimal and maximal pheromone values allowed, respectively τ_{min} and τ_{max}, Stützle and Hoos suggest that they should be chosen experimentally based on the problem at hand. The maximum value τ_{max} may be calculated analytically provided that the optimum ant tour length is known. In the case of the TSP, τ_{max} is given by

$$\tau_{max} = \frac{1}{\rho}\cdot\frac{1}{L^{*}} \tag{26.12}$$

where L^{*} is the length of the optimal tour. If L^{*} is not known, it can be approximated by L_{bs}. The minimum pheromone value τ_{min} should be chosen with caution as it has a rather strong influence on the algorithm performance. Stützle and Hoos present an analytical approach to finding this value based on the probability p_{best} that an ant constructs the best tour found so far. This is done as follows. First, it is assumed that at each construction step an ant has a constant number k of options available. Therefore, the probability that an ant makes the *right* decision (i.e., the decision that belongs to the sequence of decisions leading to the construction of the best tour found so far) at each of n steps is given by $p_{\text{dec}} = \sqrt[n-1]{p_{\text{best}}}$. The analytical formula they suggest for finding τ_{min} is

$$\tau_{min} = \frac{\tau_{max} \cdot (1 - p_{dec})}{k \cdot p_{dec}} \tag{26.13}$$

For more details on how to choose τ_{max} and τ_{min}, we refer to Ref. [16]. It is important to mention here that it has also been shown [19] that for some problems the choice of an appropriate τ_{min} value is more easily done experimentally than analytically.

The process of pheromone update in $\mathcal{MM}$AS is concluded by verifying that all pheromone values are within the imposed limits:

$$\tau_{ij} = \begin{cases} \tau_{min} & \text{if } \tau_{ij} < \tau_{min} \\ \tau_{ij} & \text{if } \tau_{min} \leq \tau_{ij} \leq \tau_{max} \\ \tau_{max} & \text{if } \tau_{ij} > \tau_{max} \end{cases} \tag{26.14}$$

$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System provided a significant improvement over the basic AS performance. While the first implementations focused on the TSP [16], it has been later applied to many other COPs such as the QAP [27], the UCTP [19], the generalized assignment problem (GAP) [28], and the set-covering problem (SCP) [29].

26.4.3 Ant Colony System

Another improvement over the original AS was Ant Colony System (ACS), introduced by Gambardella and Dorigo [15,30]. The most interesting contribution of ACS is the introduction of a *local pheromone update* in addition to the pheromone update performed at the end of the construction process (called here *offline* pheromone update).

The local pheromone update is performed by all the ants after each construction step. Each ant applies it only to the last edge traversed:

$$\tau_{ij} = (1 - \varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0 \tag{26.15}$$

where $\varphi \in (0, 1)$ is the pheromone decay coefficient, and τ_0 is the initial value of the pheromone.

The main goal of the local update is to diversify the search performed by subsequent ants during one iteration. In fact, decreasing the pheromone concentration on the edges as they are traversed during one iteration encourages subsequent ants to choose other edges and hence to produce different solutions. This makes less likely that several ants produce identical solutions during one iteration.

The offline pheromone update, similarly to $\mathcal{MM}$AS, is applied at the end of each iteration by only one ant (either the one that found the best solution in the iteration or the best-so-far). However, the update formula is slightly different:

$$\tau_{ij} \leftarrow \begin{cases} (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij} & \text{if edge } (i,\ j) \text{ belongs to the tour of the best ant} \\ \tau_{ij} & \text{otherwise} \end{cases} \tag{26.16}$$

and in case of the TSP, $\Delta\tau_{ij} = \frac{1}{L_{\text{best}}}$ (as in $\mathcal{MM}$AS, L_{best} can be set to either L_{ib} or L_{bs}.)

Another important difference between AS and ACS is in the decision rule used by the ants during the construction process. Ants in ACS use the so-called *pseudorandom proportional* rule: the probability for an ant to move from city i to city j depends on a random variable q uniformly distributed over $[0, 1]$, and a parameter q_0; if $q \leq q_0$, then $j = \text{argmax}_{l \in N(s^p)}\{\tau_{il}\eta_{il}^{\beta}\}$, otherwise Eq. (26.8) is used.

Ant Colony System has been initially developed for the TSP [15,30], but it was later used to tackle various COPs, including vehicle routing [31], sequential ordering [32], and timetabling [33].

26.5 Current Research Directions

Research in ACO is very active. It includes the application of ACO algorithms to new real-world optimization problems or new types of problems, such as dynamic optimization [34], multiobjective optimization [35], stochastic problems [36], or continuous and mixed-variable optimization [37,42]. Also, with an increasing popularity of parallel hardware architectures (multicore processors and the grid technology), a lot of research is being done on creating parallel implementations of ACO that will be able to take advantage of the available hardware. In this section we briefly present current research in these new areas.

26.5.1 Other Types of Problems

One of the new areas of application of ACO is dynamic optimization. This type of problems is characterized by the fact that the search space dynamically changes. While an algorithm searches for good solutions, the conditions of the search as well as the quality of the solutions already found may change. This poses a whole new set of issues for designing successful algorithms that can deal with such situations. It becomes crucial for an algorithm to be able to adjust the search direction, following the changes of the problem being solved. Initial attempts to apply ACO to dynamic optimization problems have been quite successful [34,38,39].

Multiobjective optimization is another area of application for metaheuristics that has received increasing attention over the past years. A multiobjective optimization problem involves solving simultaneously several optimization problems with potentially conflicting objectives. For each of the objectives, a different objective function is used to assess the quality of the solutions found. Algorithms usually aim at finding the so-called *Pareto set*, that is, a set of nondominated solutions, based on the defined objective functions. In the Pareto set, no solution is worse than any other in the set, when evaluated over all the objective functions. Some ACO algorithms designed to tackle multiobjective problems have been proposed in the literature [35,40,41].

Finally, recently researchers attempted to apply ACO algorithms to continuous optimization problems. When an algorithm designed for combinatorial optimization is used to tackle a continuous problem, the simplest approach is to divide the domain of each variable into a set of intervals. The set of intervals is finite and may be handled by the original discrete optimization algorithm. However, when the domain of the variables is large, and the required accuracy is high, this approach runs into problems. The problem size (i.e., the number of intervals) grows, and combinatorial optimization algorithms become less efficient. Also, this approach requires setting the number of intervals *a priori*—before the algorithm is run. In case of real-world problems, this is not always a sensible thing to do.

Due to these reasons, optimization algorithms able to handle continuous parameters natively have been developed. Recently, Socha and Dorigo [37,42] have extended ACO to continuous (and mixed-variable—continuous and discrete) problems. Research in this respect is ongoing and should result in new, efficient ACO implementations for continuous and mixed-variable problems.

26.5.2 Parallel ACO Implementations

Parallelization of algorithms becomes more and more an interesting and practical option for algorithm designers. Ant colony optimization is particularly well suited for parallel implementations, thanks to ants operating in an independent and asynchronous way. There have already been many attempts to propose parallel ACO algorithms. They are usually classified by their *parallel grain*, that is, the relationship between computation and communication. We can then distinguish between *coarse-* and *fine-grained* models. While the formers are characterized by many ants using the same CPU and rare communication between the CPUs, in the latters only few ants use each CPU and there is a lot of communication going on. An overview of the trends and strategies in designing parallel metaheuristics may be found in Refs. [43,44].

Randall and Lewis [45] proposed a first reasonably complete classification of parallel ACO implementations. Although many parallel ACO implementations have been proposed in the literature [46–51], the results are fragmented and difficult to compare. Experiments are usually of limited scale and concern different optimization problems. Also, not all parallel implementations proposed are compared with their sequential counterparts, which is an essential measure of their usefulness [51]. All this implies that more research is necessary in the area of parallelization of the ACO metaheuristic (for some recent work in this direction see Ref. [52]).

26.6 Conclusions

We have presented an introduction to ACO—a metaheuristic inspired by the foraging behavior of real ants. The central component of ACO is the pheromone model based on the underlying model of the problem being solved. The basic idea of ACO, which has been formalized into a metaheuristic framework, leaves many options and choices to the algorithm designer. Several variants of ACO have been already proposed, the most successful being $\mathcal{MM}$AS and ACS.

Ant colony optimization is a relatively young metaheuristic, when compared to others such as evolutionary computation, tabu search, or simulated annealing. Yet, it has proven to be quite efficient and flexible. Ant colony optimization algorithms are currently state-of-the-art for solving many COPs including the sequential ordering problem [32], the resource constraint project scheduling problem [53], and the open-shop scheduling problem [54]. For an in-depth overview of ACO, including theory and applications, the interested reader should refer to Ref. [55].

Acknowledgments

Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a research director. This work was supported by the "ANTS" project, an "Action de Recherche Concertée" funded by the Scientific Research Directorate of the French Community of Belgium.

References

[1] Dorigo, M., Maniezzo, V., and Colorni, A., Positive Feedback as a Search Strategy, Technical report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.

[2] Dorigo, M., Optimization, Learning and Natural Algorithms, Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, 1992 (in Italian).

[3] Dorigo, M., Maniezzo, V., and Colorni, A., Ant System: Optimization by a colony of cooperating agents, *IEEE Trans. Syst., Man, and Cybern. – Part B*, 26(1), 29, 1996.

[4] Camazine, S., Deneubourg, J.-L., Franks, N., Sneyd, J., Theraulaz, G., and Bonabeau, E., *Self-Organization in Biological Systems*, Princeton University Press, Princeton, NJ, 2003.

[5] Bonabeau, E., Dorigo, M., and Theraulaz, G., *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, NY, 1999.

[6] Grassé, P.-P., Recherches sur la biologie des termites champignonnistes (*Macrotermitina*), *Ann. Sc. Nat., Zool. Biol. anim.*, 6, 97, 1944.

[7] Grassé, P.-P., La reconstruction du nid et les coordinations interindividuelles chez *bellicositermes natalensis* et *cubitermes* sp. La théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs, *Insectes Sociaux*, 6, 41, 1959.

[8] Deneubourg, J.-L., Aron, S., Goss, S., and Pasteels, J.-M., The self-organizing exploratory pattern of the Argentine ant, *J. Insect Behav.*, 3, 159, 1990.

[9] Goss, S., Aron, S., Deneubourg, J.-L., and Pasteels, J.-M., Self-organized shortcuts in the Argentine ant, *Naturwissenschaften*, 76, 579, 1989.

[10] Pasteels, J. M., Deneubourg, J.-L., and Goss, S., Self-organization mechanisms in ant societies (i): Trail recruitment to newly discovered food sources, *Experientia Supplementum*, 54, 155, 1987.

[11] Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, MI, 1975.

[12] Fogel, D. B., *Evolutionary Computation*, IEEE Press, Piscataway, NJ, 1995.

[13] Dorigo, M. and Di Caro, G., The ant colony optimization meta-heuristic, in *New Ideas in Optimization*, Corne, D., Dorigo, M., and Glover, F., Eds., McGraw-Hill, New York, 1999, p. 11.

[14] Dorigo, M., Di Caro, G., and Gambardella, L. M., Ant algorithms for discrete optimization, *Artif. Life*, 5(2), 137, 1999.

[15] Dorigo, M. and Gambardella, L. M., Ant Colony System: a cooperative learning approach to the traveling salesman problem, *IEEE Trans. Evol. Comput.*, 1(1), 53, 1997.

[16] Stützle, T. and Hoos, H. H., $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System, *Future Generation Comput. Syst.*, 16(8), 889, 2000.

[17] Lawler, E. L., Lenstra, J. K., Rinnooy-Kan, A. H. G., and Shmoys, D. B., *The Travelling Salesman Problem*, Wiley, New York, NY, 1985.

[18] Stützle, T. and Dorigo, M., ACO algorithms for the traveling salesman problem, in *Evolutionary Algorithms in Engineering and Computer Science*, Miettinen, K., Mäkelä, M. M., Neittaanmäki, P., and Périaux, J., Eds., Wiley, New York, 1999, p. 163.

[19] Socha, K., Knowles, J., and Sampels, M., A $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the university timetabling problem, *Proc. ANTS 2002 – 3rd Int. Workshop on Ant Algorithms*, Lecture Notes in Computer Science, Vol. 2463, Springer, Berlin, 2002, p. 1.

[20] Maniezzo, V., Colorni, A., and Dorigo, M., The Ant System Applied to the Quadratic Assignment Problem, Technical report IRIDIA/94-28, IRIDIA, Université Libre de Bruxelles, Belgium, 1994.

[21] Maniezzo, V. and Colorni, A., The Ant System applied to the quadratic assignment problem, *IEEE Trans. Knowl. Data Eng.*, 11(5), 769, 1999.

[22] Colorni, A., Dorigo, M., Maniezzo, V., and Trubian, M., Ant System for job-shop scheduling, *JORBEL—Belgian J. Operations Res., Stat. Comput. Sci.*, 34(1), 39, 1994.

[23] Bullnheimer, B., Hartl, R. F., and Strauss, C., Applying the Ant System to the vehicle routing problem, in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Osman, I. H., Voß, S., Martello, S., and Roucairol, C., Eds., Kluwer Academic Publishers, Dordrecht, 1998, p. 109.

[24] Bullnheimer, B., Hartl, R. F., and Strauss, C., An improved Ant System algorithm for the vehicle routing problem, *Ann. Oper. Res.*, 89, 312, 1999.

[25] Michel, R. and Middendorf, M., An island model based Ant System with lookahead for the shortest supersequence problem, *Proc. PPSN-V, 5th Int. Conf. on Parallel Problem Solving from Nature*, Springer, Berlin, 1998, p. 692.

[26] Michel, R. and Middendorf, M., An ACO algorithm for the shortest common supersequence problem, in *New Ideas in Optimisation*, Corne, D., Dorigo, M., and Glover, F., Eds., McGraw-Hill, Boston, MA, 1999, p. 51.

[27] Stützle, T. and Hoos, H., The $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System and local search for combinatorial optimization problems: towards adaptive tools for combinatorial global optimisation, in *Meta-Heuristic, Advances and Trends in Local Search Paradigms for Optimization*, Voss, S., Martello, S., Osman, I. H., and Roucairol, C., Eds., Kluwer Academic Publishers, Dordrecht, 1998, p. 313.

[28] Lourenço, H. R. and Serra, D., Adaptive approach heuristics for the generalized assignment problem, TR Economic Working Papers Series No. 304, Universitat Pompeu Fabra, Department of Economics and Management, Barcelona, Spain, 1998.

[29] Lessing, L., Dumitrescu, I., and Stützle. T., A comparison between ACO algorithms for the set covering problem, *Proc. 4th Int. Workshop on Ant Algorithms and Swarm Intelligence*, Lecture Notes in Computer Science, Vol. 3172, Springer, Berlin, 2004, p. 1.

[30] Gambardella, L. M. and Dorigo, M., Solving symmetric and asymmetric TSPs by ant colonies, *Proc. '96 IEEE Int. Conf. on Evolutionary Computation (ICEC'96)*, IEEE Press, New York, 1996, p. 622.

[31] Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O., and Schiavinotto, T., Metaheuristics for the vehicle routing problem with stochastic demands, *Proc. Parallel Problem Solving from Nature—PPSN VIII, 8th International Conference*, Lecture Notes in Computer Science, Vol. 3242, Springer, Berlin, 2004, p. 450.

[32] Gambardella, L. M. and Dorigo, M., Ant Colony System hybridized with a new local search for the sequential ordering problem, *INFORMS J. Comput.*, 12(3), 237, 2000.

[33] Socha, K., Sampels, M., and Manfrin, M., Ant algorithms for the university course timetabling problem with regard to the state-of-the-art, *Proc. EvoCOP 2003 – 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization*, Lecture Notes in Computer Science, Vol. 2611, Springer, Berlin, 2003, p. 334.

[34] Guntsch, M. and Middendorf, M., Applying population based ACO to dynamic optimization problems, *Proc. ANTS 2002—3rd Int. Workshop on Ant Algorithms*, Lecture Notes in Computer Science, Vol. 2463, Springer, Berlin, 2002, p. 111.

[35] Iredi, S., Merkle, D., and Middendorf, M., Bi-criterion optimization with multi colony ant algorithms, *Proc. Evolutionary Multi-Criterion Optimization, 1st Int. Conf. (EMO'01)*, Lecture Notes in Computer Science, 1993, Springer, Berlin, 2001, p. 359.

[36] Gutjahr, W. J., S-ACO: An ant-based approach to combinatorial optimization under uncertainty, *Proc. 4th Int. Workshop on Ant Algorithms and Swarm Intelligence*, Lecture Notes in Computer Science, Vol. 3172, Springer, Berlin, 2004, p. 238.

[37] Socha, K., ACO for continuous and mixed-variable optimization, *Proc. Ant Colony Optimization and Swarm Intelligence, 4th Int. Workshop, ANTS 2004*, Lecture Notes in Computer Science, Vol. 3172, Springer, Berlin, 2004, p. 25.

[38] Di Caro, G., and Dorigo, M., AntNet: distributed stigmergetic control for communications networks, *J. Artif. Intell. Res. (JAIR)*, 9, 317, 1998.

[39] Guntsch, M., Middendorf, M., and Schmeck, H., An ant colony optimization approach to dynamic TSP, *Proc. Genetic and Evolutionary Computation Conference (GECCO-2001)*, Morgan Kaufmann, San Francisco, CA, 2001, p. 860.

[40] Guntsch, M. and Middendorf, M., Solving multi-criteria optimization problems with population-based ACO, *Proc. of Evolutionary Multi-Criterion Optimization: 2nd Int. Conf. EMO 2003*, Lecture Notes in Computer Science, Vol. 2632, Springer, Berlin, 2003, p. 464.

[41] Doerner, K., Gutjahr, W., Hartl, R., Strauss, C., and Stummer, C., Pareto ant colony optimization: a metaheuristic approach to multiobjective portfolio selection, *Ann. Oper. Res.*, 131(1–4), 79, 2004.

[42] Socha, K. and Dorigo, M., Ant Colony Optimization for Continuous Domains, *European Journal of Operational Research*, doi:10.1016/j.ejor.2006.06.046, in press.

[43] Cung, V.-D., Martins, S. L., Ribeiro, C. C., and Roucairol, C., Strategies for the parallel implementation of metaheuristics, in *Essays and Surveys in Metaheuristics*, Ribeiro, C. C. and Hansen, P., Eds., Operations Research/Computer Science Interfaces, Vol. 15, Kluwer Academic Publishers, Dordrecht, 2001, chap. 13.

[44] Alba, E., *Parallel Metaheuristics. A New Class of Algorithms*, Wiley, Cambridge, NJ, 2005.

[45] Randall, M. and Lewis, A., A parallel implementation of ant colony optimization, *J. Parallel Distributed Comput.*, 62(9), 1421, 2002.

[46] Merkle, D. and Middendorf, M., Fast ant colony optimization on runtime reconfigurable processor arrays, *Genet. Programming and Evolvable Machines*, 3(4), 345, 2002.

[47] Talbi, E.-G., Roux, O., Fonlupt, C., and Robillard, D., Parallel ant colonies for combinatorial optimization problems, *Proc. 11th IPPS/SPDP'99 Workshops held in conjunction with the 13th International Parallel Processing Symp. and the 10th Symp. on Parallel and Distributed Processing*, Springer, Berlin, 1999, p. 239.

[48] Gambardella, L. M., Taillard, E., and Agazzi, G., MACS-VRPTW: A multiple Ant Colony System for vehicle routing problems with time windows, in *New Ideas in Optimization*, Corne, D., Dorigo, M., and Glover, F., Eds., McGraw-Hill, New York, 1999, p. 63.

[49] Rahoual, M., Hadji, R., and Bachelet, V., Parallel Ant System for the set covering problem, *Proc. Ant Algorithms—3rd Int. Workshop, ANTS 2002*, Lecture Notes in Computer Science, Vol. 2463, Springer, Berlin, 2002, p. 262.

[50] Bullnheimer, B., Kotsis, G., and Strauß, G., Parallelization strategies for the Ant System, Technical report 8, Vienna University of Economics and Business Administration, Vienna, Austria, 1997.

[51] Stützle, T., Parallelization strategies for ant colony optimization, *Proc. Parallel Problem Solving from Nature—PPSN V: 5th Int. Conf.*, LNCS, 1498, Springer, Berlin, 1998, p. 722.

[52] Manfrin, M., Birattari, M., Stützle, T., and Dorigo, M., Parallel Ant Colony Optimization for the Traveling Salesman Problem, Technical report TR/IRIDIA/2006-007, IRIDIA, Université Libre de Bruxelles, Belgium, 2006.

[53] Merkle, D., Middendorf, M., and Schmeck, H., Ant colony optimization for resource-constrained project scheduling, *IEEE Trans. Evol. Comput.*, 6(4), 333, 2002.

[54] Blum, C., Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling, *Comput. Oper. Res.*, 32(6), 1565, 2005.

[55] Dorigo, M. and Stützle, T., *Ant Colony Optimization*, MIT Press, Cambridge, MA, 2004.

27

Memetic Algorithms

Pablo Moscato
The University of Newcastle

Carlos Cotta
University of Málaga

27.1 Introduction

The term *memetic algorithms* [1] (MAs) was introduced in the late 1980s to denote a family of metaheuristics that have as central theme the hybridization of different algorithmic approaches for a given problem. Special emphasis was given to the use of a population-based approach in which a set of cooperating and competing agents was engaged in periods of individual improvement of the solutions while they interact sporadically. Another main objective theme was to introduce *problem and instance-dependent knowledge* as a way of speedingup the search process. Initially, hybridizations included Evolutionary Algorithms (EAs) [2–5], Simulated Annealing and its variants [6,7], and Tabu Search [8,9]. Today, a number of hybridizations include other metaheuristics [10] as well as exact algorithms, in *complete anytime* memetic algorithms [11]. These methods not only prove optimality, but can also deliver high-quality solutions early on in the process.

The adjective "memetic" comes from the term "meme," coined by R. Dawkins [12], to denote a term analogous to the *gene* in the context of cultural evolution. It was first proposed as a means of conveying the message that, although inspiring for many, biological evolution should not constrain the imagination to develop population-based methods. Other forms of evolution may be faster, with cultural evolution being one of those less-restrictive examples.

Due to the fact that MAs aimed at drawing the attention from two communities of researchers with different agendas, focusing at hybridizations of their methods, this metaheuristic had to suffer tough initial times. Today they are becoming increasingly popular due to their impressive success record and the high sophistication of the hybridizations proposed. As a consequence of its evolution, it is not unusual to find MAs disguised in the literature under different names such as "hybrid EAs" or "Lamarckian EAs." Furthermore, many of the underlying ideas of MAs can also be found in other metaheuristics that evolved in relative isolation from MAs. Scatter search [13] is a good example of a metaheuristic sharing much of its functioning philosophy with MAs. In Ref. [14], the authors cite a paper by S. Kase, in which a "game" between a set of hierarchical agents (players and referees) is proposed to hybridize heuristic approaches for a layout problem [15]. What makes this interesting is that this is an approach that does not rely on computers for optimization and helps the employees to become engaged in these issues. Anticipating further definitions, we can say that an MA is a search strategy in which a population of optimizing agents synergistically cooperate and compete [16]. A more detailed description of the algorithm as well as a functional template will be given in Section 27.2.

As mentioned before, MAs are a hot topic nowadays, mainly due to their success in solving many hard optimization problems, attracting experienced researchers to work on the challenges of this field. A particular feature of MAs accounts for this: unlike traditional EAs, MAs are intrinsically concerned with exploiting *all available knowledge* about the problem under study. The advantages of this approach were notably neglected in EAs for a long time despite some contrary voices, most notably Davis' who also advocated for hybridization in his book [17]. The formulation of the so-called *No-Free-Lunch Theorem* (NFL) by Wolpert and Macready [18] made it definitely clear that a search algorithm strictly performs in accordance with the amount and quality of the problem knowledge they incorporate, thus backing up one of the *leiv motivs* of MAs.

MAs exploit problem knowledge by incorporating preexisting heuristics, preprocessing data reduction rules, approximation and fixed-parameter tractable algorithms, local search techniques, specialized recombination operators, truncated exact methods, and so on. Also, an important factor is the use of adequate representations of the problem being tackled. This results in highly efficient optimization tools. We provide a brief abstracted overview of MA applications in combinatorial optimization in Section 27.3. We will finish with a brief summary of the current research trends in MAs, with special mention to those which we believe will play a major role in the near future.

27.2 Dissecting a Basic Memetic Algorithm

As mentioned in the previous section, MAs blend different search strategies in a combined algorithmic approach. Like EAs, MAs are population-based metaheuristics. This means that in MAs we maintain a *population* of solutions for the problem at hand. We are using the term "solutions" rather loosely here, as we can have either feasible or protosolutions (structures that can be extended/modified to produce feasible solutions) or even unfeasible solutions (which can be "repaired" to restore feasibility). It is also assumed that both repairing and extension processes can be done quite fast, as to justify including them in the population. Each of these solutions will be termed *individual* as in the EA jargon, mainly to simplify the discussion. In the context of MAs, the denomination *agent* represents a processing unit that can hold multiple solutions and has problem-domain methods that help to improve them if required [1]. Each individual/agent represents a tentative solution/method for the problem under consideration. When the agents adapt their methods we call the resulting strategy an *adaptive memetic algorithm*. Adaptation may include a modification of the data as in Ref. [10].

Due to the agent's interactions, solutions are subject to processes of competition and mutual cooperation. The general structure of MAs is shown in Figure 27.1, aiming to highlight similarities with other population-based metaheuristics such as EAs. Relevant differences are nevertheless evident when we inspect the innards of the high-level blocks depicted in Figure 27.1. First of all, notice the existence of an

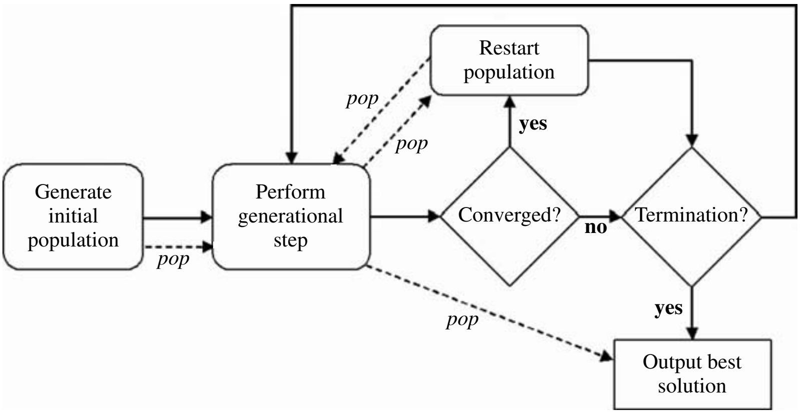


FIGURE 27.1 The general structure of MAs. Solid arrows indicate the control flow, whereas broken arrows indicate the data flow.

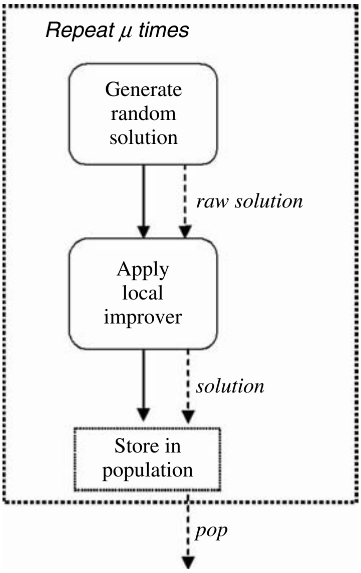


FIGURE 27.2 Generation of the initial population. A local improver can be used to enhance the quality of starting solutions.

initialization block. Standard EAs would simply generate $\mu = |pop|$ random solutions. This can be also done in MAs, but more sophisticated mechanisms are typically used as they are more useful. For example, some constructive heuristic can be utilized to produce high-quality solutions [19,20]. Another possibility refers to the use of a local improvement method, as illustrated in Figure 27.2.

There is another interesting element in the flow chart shown in Figure 27.1: the *Re-start Population* process. This component is sometimes present in some EAs, but it is essential in MAs. Consider that the population may reach a state in which the generation of a new improved solution might be very unlikely. This could be the case when all solutions in the population are very similar to each other. In this situation of population convergence, it is better to refresh the population, rather than keeping the population constrained to a small region of the search space, probably expending most of the time resampling the same solutions [21]. This is specifically important in MAs since the inclusion of several knowledge-augmented components contributes to accelerate the convergence of the population. Typical criteria for determining population convergence are measuring the diversity of solutions—via Shannon's entropy [22] for instance—and Bayesian decision making [23]. In either case, and whenever the population is considered to have converged, restarting can be done in different ways. One of these is shown in Figure 27.3: top individuals of the population are kept (a certain fraction p of the population; this value should not be very high since otherwise the population would obviously converge again in a very short time afterwards), and the remaining solutions are created from scratch, as it is done in the initialization phase.

The main functional block in the MA template is the *generational step* process. This is actually the part of the algorithm in which evolution of solutions takes place. Its internal structure is depicted in Figure 27.4. As can be seen, there are three main components in this generational step: selection, reproduction, and update. Selection and update are responsible for the competition aspects of individuals in the population. Using the information provided by a problem-dependent guiding function (termed *fitness* function in the EA terminology), the goodness of individuals in pop is evaluated, and a sample of individuals is selected according to this goodness measure to help create new solutions. Essentially, this selection can be done using fitness-proportionate methods (the probability of selecting an individual is proportional to its fitness) and nonproportionate methods (selection is done on the basis of qualitative comparisons

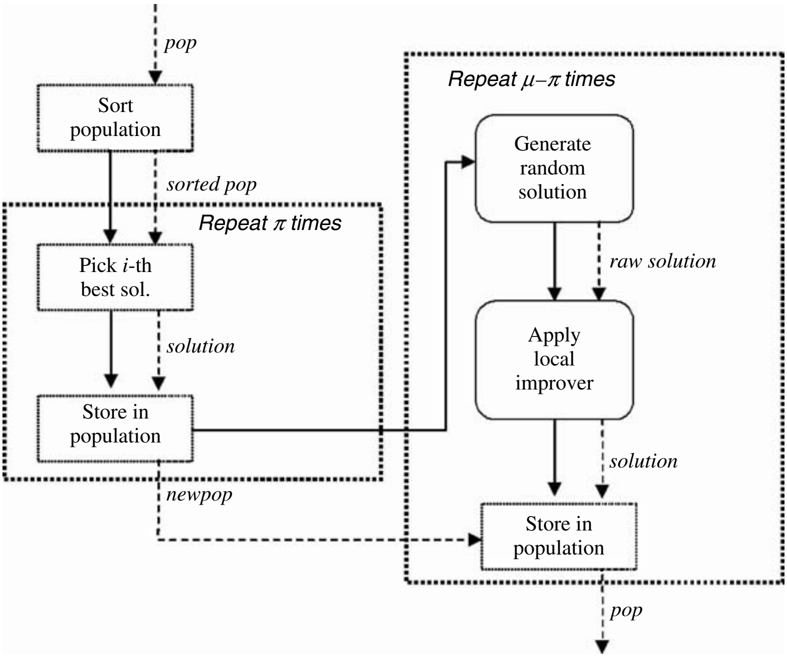


FIGURE 27.3 A possible restarting procedure for the population. The top $\pi = p\mu$ agents in the population are kept, and the remaining μ–π are generated from scratch.

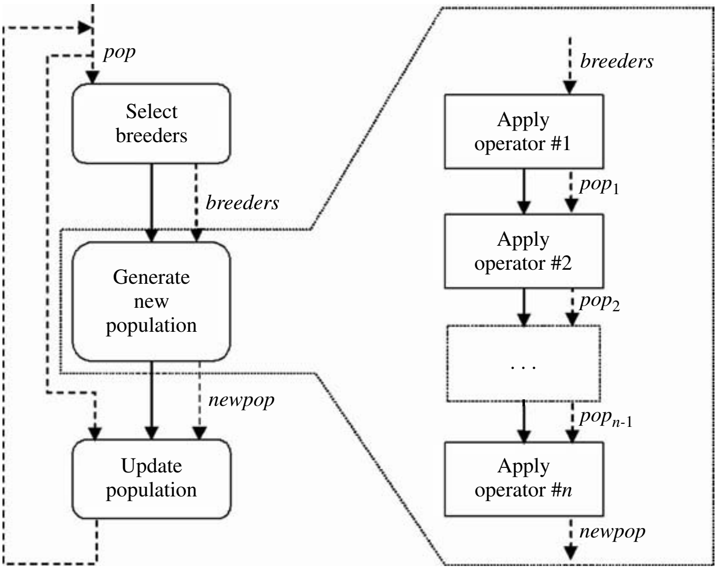


FIGURE 27.4 The basic generational step. Notice the use of a pipeline of reproductive operators for creating new solutions.

among individuals). The latter are being increasingly used, since they avoid some problems of the former (assumption of maximization, need of transformation for dealing with minimization, scaling problems, etc.). Among these, we can cite rank- (the top in the rank of an individual, the higher are its chances for being selected) and tournament-based methods (individuals are selected on the basis of a direct competition within small subgroups of individuals).

As to update, this component takes care of maintaining the population at a constant size, or more properly, at a manageable size, since variable-size populations are not rare [24]. This is done by substituting some preexisting individuals in *pop* by some of the new ones from *newpop*, using some specific criteria. Two major strategies are possible: the *plus* strategy in which the best μ individuals from $pop \cup newpop$ are kept and the *comma* strategy in which the best μ from *newpop* are kept. In the latter case, if $|pop| = |newpop|$ then the update is termed *generational*; if $|newpop|$ is small (say $|newpop| = 1$) then we have a *steady-state* replacement (the worst $|newpop|$ solutions from *pop* are substituted). Other intermediate *generational gaps* are possible by selecting higher values of $|newpop|$.

We finally arrive at the reproduction stage, where new individuals (or agents) are created using the information existing in the population. More precisely, several reproductive *operators* (i.e., transformation functions) are used in a pipelined fashion, as illustrated in Figure 27.4. Reproductive operators are algorithms that can be classified into two classes: unary operators and n-ary ($n > 1$) operators. Beginning with the former, two further types of operators are typically used, namely *mutation* operators and *individual-improvement* operators (in many cases based on some form of local search). The latter were already mentioned before, for example, in the initialization phase. As indicated by their name, their purpose is to improve the quality of a certain solution. In general, this is implemented via an iterative process whereby small modifications are introduced in a solution and kept if they result in an effective quality improvement. This process is repeated until it can be determined that no further improvement is possible, until the amount of quality improvement is considered good enough, or—most typically—until a maximum number of improving attempts are performed. Hence, the process need not stop at an optimum for the individual improver, and therefore characterizations of MAs as *"EAs working in the space of local-optima [with respect to a certain fitness landscape]"* are clearly restricting even the methods that originated the denomination [1,25] and should be avoided. As to mutation, it is intended to generate new solutions by partly modifying existing solutions. This modification can be random—as it is typically the case in EAs—or can be endowed with problem-dependent information so as to bias the search to probably good regions of the search space.

Nonunary operators are usually termed recombination operators. These operators constitute a distinctive added-value possibility of population-based search and encapsulate the mutual cooperation among several individuals (typically two of them, but a higher number is possible). They generate new individuals using the information contained in a number of selected solutions called *parents.* If the resulting individuals (the *offspring*) are entirely composed of information taken from the parents, then the recombination is said to be *transmitting* [26]. This property can be difficult to achieve for certain problem domains (the *Traveling Salesman Problem* (TSP) is a typical example). In those situations, it is possible to consider other properties of interest such as *respect* or *assortment.* The former refers to the fact that the recombination operator generates descendants carrying all *features* (i.e., basic properties of solutions with relevance for the problem attacked) common to all parents; thus, this property can be seen as a part of the *exploitative* side of the search. In contrast, *assortment* represents the exploratory side of recombination. A recombination operator is said to be *assorting* if, and only if, it can generate descendants carrying any combination of compatible features taken from the parents. In either case, similar to mutation, performing the combination of information in a problem-oriented way (rather than blindly) is crucial for the performance of the algorithm (see, e.g., Refs. [27,28]).

This description of recombination has introduced a crucial concept, namely, *relevant features.* By relevant features we mean the information pieces that can be extracted from solutions, exerting a direct influence on the quality of these. Consider that a certain solution can contain a high number of *atomic* information units, but only some of them are directly linked with quality. For example, a permutation π can be interpreted as a collection of positional information units, that is, position i has value π_i. It also can be interpreted as a collection of adjacency information units, that is, values a and b occur in adjacent positions of the permutation. It turns out that if the permutation is taken as a solution to the TRAVELING SALESMAN PROBLEM, the latter are indeed the relevant features, while for the FLOWSHOP SCHEDULING PROBLEM positional information is much more important [29]. The definition of operators manipulating the relevant features is one of the key aspects in the design of MAs.

There have been several attempts for quantifying how good a certain set of information units is for representing solutions for a specific problems. We can cite a few of them.

- *Minimizing epistasis.* Epistasis can be defined as the nonadditive influence on the guiding function of combining several information units (see, e.g., Ref. [30]). Clearly, the higher this nonadditive influence, the lower the absolute relevance of individual information units. Since the algorithm will be processing such individual units (or small groups of them), the guiding function turns out to be low informative, and prone to misguide the search.
- *Minimizing fitness variance.* [26]. This criterion is strongly related to the previous one. The fitness variance for a certain information unit is the variance of the values returned by the guiding function, measured across a representative subset of solutions carrying this information unit. By minimizing this fitness variance, the information provided by the guiding function is less *noisy*, with the subsequent advantages for the guidance of the algorithm.
- *Maximizing fitness correlation.* In this case a certain reproductive operator is assumed, and the correlation in the values of the guiding function for parents and offspring is measured. If the fitness correlation is high, good solutions are likely to produce good solutions too, and thus the search will gradually shift toward the most promising regions of the search space. Again, there is a clear relationship with the previous approaches; for instance, if epistasis (or fitness variance) is low, then solutions carrying specific features will have similar values for the guiding function; since the reproductive operators will create new solutions by manipulating these features, the offspring is likely to have a similar guiding value as well.

Obviously, the description of these approaches may appear somewhat idealized, but the underlying philosophy is well illustrated. For further advice on the design of MAs, the reader is referred to Refs. [31,32].

27.3 MAs and Combinatorial Optimization

MAs constitute a extremely powerful tool for tackling combinatorial optimization problems. Indeed, MAs are state-of-the-art approaches for many such problems. Traditional NP optimization problems constitute one of the most typical battlefields of MAs, and a remarkable history of successes has been reported with respect to the application of MAs to such problems. Combinatorial optimization problems (both single- and multiobjective [33–35]) arising in scheduling, manufacturing, telecommunications, and bioinformatics among other fields have been also satisfactorily tackled with MAs. Some of these applications are summarized in Table 27.1.

This list of applications is by no means complete since its purpose is simply to document the wide applicability of the approach for combinatorial optimization. Indeed, MAs have been successfully applied to other domains. Other such application areas of MAs include machine learning, robotics, engineering, electronics, bioinformatics, and oceanography. For further information about MA applications we suggest checking Refs. [31,32], or querying bibliographical databases or web browsers for the keywords "*memetic algorithms*" and "*hybrid genetic algorithm*."

27.4 Conclusions and Future Directions

We believe that MAs have very favorable perspectives for their development and widespread application. We ground our belief for several reasons: first, MAs are showing a great record of efficient implementations, providing very good results in practical problems (cf. previous section). We also have reasons to believe that we are near some major leaps forward in our theoretical understanding of these techniques, including, for example, the computational complexity analysis of recombination procedures. In contrast, the inherent asynchronous parallelism of MAs adapts very well to the increasing availability of distributed systems.

TABLE 27.1 Some Applications of Memetic Algorithms in Combinatorial Optimization

Graph Partitioning	[40,41]	Min Number Partitioning	[9,42]
Max Independent Set	[43–45]	Bin-Packing	[46]
Min Graph Coloring	[47,48]	Set Covering	[49]
Min Generalized Assignment	[50]	Multidimensional Knapsack	[51–53]
Quadratic Assignment	[54,55]	Quadratic Programming	[56]
Set Partitioning	[57]	Gate Matrix Layout	[58,59]
Traveling Salesman Problem	[10,60,61,63–65]	Min Weighted k-Cardinality Tree	[62]
Uncapacitated Hub Location	[67]	Min k-Cut Problem	[66]
Vehicle Routing	[71–73]	Placement Problems	[68–70]
Prize-Collecting Steiner Tree	[75]	Task Allocation	[74]
Vertex-Biconnectivity Augmentation	[79]	Network Design	[76–78]
		Error Correcting Codes	[80]
OSPF Routing	[81]	Maintenance Scheduling	[82]
Open Shop Scheduling	[83]	Flowshop Scheduling	[84–86]
Single Machine Scheduling	[87–89]	Parallel Machine Scheduling	[90]
Project Scheduling	[91]	Production Planning	[92]
Timetabling	[93,94]	Rostering	[95]
Sport Games Scheduling	[96,97]	Airport Gate Scheduling	[98,99]
Multistage Capacitated Lot-Sizing	[100]	Graph Isomorphism Problem	[101]
Protein Structure Prediction	[36,102,103]	Clustering	[104–106]

We also see as a healthy sign the systematic development of other particular optimization strategies. If a simpler, nonpopulation-based, metaheuristic performs the same as a more complex method (GAs, MAs, Ant Colonies, and so on), Ockham's razor should prevail and we must resort either to the simpler method or to the one that has less free parameters, or to the one that is easier to implement. Such a fact should defy us to adapt the complex methodology to beat a simpler heuristic, or to check if that is possible at all. An unhealthy sign of current research, however, is the attempt to encapsulate metaheuristics on stretched confinements. The evolutionary computing community had to endure a difficult time in the past, until the artificial boundaries among the different EA families were overcome. It would be unwise to repeat the same mistakes in the wider context of metaheuristics.

There are many open lines of research in MAs. One of them is multilevel evolution. It was anticipated in Ref. [11] that future MAs could simultaneously evolve solutions (in a short timescale), as well as representations and methods (in a longer timescale). In this sense, Krasnogor and collaborators have recently introduced techniques to adaptively change the neighborhood definition [36], and used these adaptive memetic algorithms for the difficult problem of *protein structure prediction* [37]. Smith [38,39] also presents a recent study on these issues.

Multiparent recombination is another promising area in which further work has to be done. Recall that recombination is precisely one of the additional search possibilities contributed by population-based algorithms, and that its augmentation with problem knowledge results in notably enhanced optimization capabilities. It seems natural to generalize these ideas to multiple-solution recombination. Not only can one have a wider pool of information for building the offspring, but additional hints can be obtained with respect to, for example, negative knowledge, that is, what pieces of information should be avoided in the offspring. This is definitely one of the most challenging issues for future development in MAs.

References

[1] Moscato, P., On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, 1989.

[2] Fogel, L. J., Owens, A. J., and Walsh, M. J., *Artificial Intelligence through Simulated Evolution*, Wiley, New York, 1966.

[3] Holland, J., *Adaptation in Natural and Artificial Systems,* University of Michigan Press, Michigan, 1975.

[4] Rechenberg, I., *Evolutionsstrategie: Optimierung Technischer Systeme Nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog, Stuttgart, 1973.

[5] Schwefel, H.-P., *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*, Diplomarbeit, Technische Universität Berlin, Hermann Föttinger–Institut für Strömungstechnik, März 1965.

[6] Kirkpatrick, S., Gelatt, C. D., Jr., and Vecchi, M. P., Optimization by simulated annealing, *Science*, 220(4598), 671, 1983.

[7] Moscato, P. and Norman, M. G., A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems, in *Parallel Computing and Transputer Applications*, Valero, M. et al., Eds., IOS Press, Amsterdam, 1992, p. 177.

[8] Moscato, P., An introduction to population approaches for optimization and hierarchical objective functions: the role of tabu search, *Ann. Oper. Res.*, 41(1–4), 85, 1993.

[9] Berretta, R., Cotta, C., and Moscato, P., Enhancing the performance of memetic algorithms by using a matching-based recombination algorithm: results on the number partitioning problem, in *Metaheuristics: Computer-Decision Making*, Resende, M. and de Sousa, J. P., Eds., Kluwer Academic Publishers, Dordrecht, 2003, p. 65.

[10] Holstein, D., and Moscato, P., Memetic algorithms using guided local search: a case study, in *New Ideas in Optimization*, Corne, D., Dorigo, M., and Glover, F., Eds., McGraw-Hill, New York, 1999, p. 235.

[11] Moscato, P., Memetic algorithms: a short introduction, in *New Ideas in Optimization*, Corne, D., Dorigo, M., and Glover, F., Eds., McGraw-Hill, New York, 1999, p. 219.

[12] Dawkins, R., *The Selfish Gene*, Clarendon Press, Oxford, 1976.

[13] Glover, F., Laguna, M., and Martí, R., Fundamentals of scatter search and path relinking, *Control Cybern.*, 39(3), 653, 2000.

[14] Moscato, P. and Tinetti, F., Blending Heuristics with a Population-Based Approach: A Memetic Algorithm for the Traveling Salesman Problem, Report 92-12, Universidad Nacional de La Plata, Argentina, 1992.

[15] Kase, S. and Nishiyama, N., An industrial engineering game model for factory layout, *J. Ind. Eng.*, XV(3), 148, 1964.

[16] Norman, M. G. and Moscato, P., A Competitive and Cooperative Approach to Complex Combinatorial Search, Technical Report Caltech Concurrent Computation Program, Report. 790, California Institute of Technology, Pasadena, CA, 1989.

[17] Davis, L., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold Computer Library, New York, 1991.

[18] Wolpert, D. H. and Macready, W. G., No free lunch theorems for optimization, *IEEE Trans. Evol. Comput.*, 1(1), 67, 1997.

[19] Surry, P. D. and Radcliffe, N. J., Inoculation to initialise evolutionary search, in *Evolutionary Computing: AISB Workshop*, Fogarty, T. C., Ed., Lecture Notes in Computer Science, Vol. 1143, Springer, Berlin, 1996, p. 269.

[20] Louis, S. J., Yin, X., and Yuan, Z. Y., Multiple vehicle routing with time windows using genetic algorithms, *Proc. Congress on Evolutionary Computation*, 1999, p. 1804.

[21] Cotta, C., On resampling in nature-inspired heuristics, in *Proc. 7th Conf. of the Spanish Association for Artificial Intelligence*, Botti, V., Ed., 1997, p. 145 (in Spanish).

[22] Davidor, Y. and Ben-Kiki, O., The interplay among the genetic algorithm operators: information theory tools used in a holistic way, in *Parallel Problem Solving From Nature II*, Männer, R. and Manderick, B., Eds., Elsevier, Amsterdam, 1992, p. 75.

[23] Hulin, M., An optimal stop criterion for genetic algorithms: A bayesian approach, *Proc. 7th Int. Conf. on Genetic Algorithms*, 1997, p. 135.

[24] Fernández, F., Vanneschi, L., and Tomassini, M., The effect of plagues in genetic programming: a study of variable-size populations, in *Genetic Programming, Proc. EuroGP'2003*, Ryan, C. et al.,

Eds., Laboratory of Cellular and Synaptic Neurophysiology, Vol. 2610, Springer, Berlin, 2003, p. 317.

[25] Moscato, P., On Genetic Crossover Operators for Relative Order Preservation, C3P Report 778, California Institute of Technology, 1989.

[26] Radcliffe, N. J. and Surry, P. D., Fitness Variance of Formae and Performance Prediction, *Proc. 3rd Workshop on Foundations of Genetic Algorithms*, 1994, p. 51.

[27] Cotta, C. and Troya, J. M., Embedding branch and bound within evolutionary algorithms, *Appl. Intell.*, 18(2), 137, 2003.

[28] Nagata, Y. and Kobayashi, Sh., Edge assembly crossover: a high-power genetic algorithm for the traveling salesman problem, *Proc. 7th Int. Conf. on Genetic Algorithms,* 1997, p. 450.

[29] Cotta, C. and Troya, J. M., Genetic forma recombination in permutation flowshop problems, *Evol. Comput.*, 6(1), 25, 1998.

[30] Davidor, Y., Epistasis variance: a viewpoint on GA-hardness, in *Foundations of Genetic Algorithms*, Rawlins, G. J. E., Ed., Morgan Kaufmann, San Mateo CA, 1991, p. 23.

[31] Moscato, P. and Cotta, C., A gentle introduction to memetic algorithms, in *Handbook of Metaheuristics*, Glover, F. and Kochenberger, G., Eds., Kluwer Academic Publishers, Dordrecht, 2003, p. 105.

[32] Moscato, P., Cotta, C., and Mendes, A., Memetic algorithms, in *New Optimization Techniques in Engineering*, Onwubolu G. C. and Babu, B. V., Eds., Springer, Berlin, 2004, p. 53.

[33] Ishibuchi, H. and Narukawa, K., Some issues on the of implementation of local search in evolutionary multi-objective optimization, in Lecture Notes in Computer Science, Vol. 3102, Springer, Berline, 2004, p. 1246.

[34] Jaszkiewicz, A., A comparative study of multiple-objective metaheuristics on the bi-objective set covering problem and the pareto memetic algorithm, *Ann. Oper. Res.*, 131(1–4), 135, 2004.

[35] Knowles, J. D. and Corne, D. W., M-paes: a memetic algorithm for multiobjective optimization, *Congress on Evolutionary Computation,* 2000, p. 325.

[36] Krasnogor, N. and J. Smith, J., Multimeme algorithms for the structure prediction and structure comparison of proteins, *GECCO 2002: Proc. Bird of a Feather Workshops*, 2002, p. 42.

[37] Krasnogor, N., Blackburne, B., Hirst, J. D., and Burke, E. K., Multimeme algorithms for protein structure prediction, *Proc. 7th Int. Conf. on Parallel Problem Solving from Nature—PPSN VII*, 2002.

[38] Smith, J. E., Protein structure prediction with co-evolving memetic algorithms, *The Congress on Evolutionary Computation*, 2003, p. 2346.

[39] Santos, E. E. and Santos Jr., E., Reducing the computational load of energy evaluations for protein folding, *Proc. 4th IEEE Symp. on Bioinformatics and Bioengineering,* 2004, p. 79.

[40] Bui, T. N. and Moon, B. R., GRCA: a hybrid genetic algorithm for circuit ratio-cut partitioning, *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.*, 17(3), 193, 1998.

[41] Merz, P. and Freisleben, B., Memetic algorithms and the fitness landscape of the graph bi-partitioning problem, *Proc. 5th Int. Conf. on Parallel Problem Solving from Nature—PPSN V*, Lecture Notes in Computer Science, Vol. 1498, Springer, Berlin, 1998, p. 765.

[42] Berretta, R. and Moscato, P., The number partitioning problem: an open challenge for evolutionary computation? *New Ideas in Optimization*, Corne, D., Dorigo, M., and Glover, F., Eds., McGraw-Hill, New York, 1999, p. 261.

[43] Aggarwal, C. C., Orlin, J. B., and Tai, R. P., Optimized crossover for the independent set problem, *Oper. Res.*, 45(2), 226, 1997.

[44] Hifi, M., A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems, *J. Oper. Res. Soc.*, 48(6), 612, 1997.

[45] Sakamoto, A., Liu, X. Z., and Shimamoto, T., A genetic approach for maximum independent set problems, *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E80A(3), 551, 1997.

[46] Reeves, C., Hybrid genetic algorithms for bin-packing and related problems, *Ann. Oper. Res.*, 63, 371, 1996.

[47] Coll, P. E., Durán, G. A., and Moscato, P., On worst-case and comparative analysis as design principles for efficient recombination operators: a graph coloring case study, in *New Ideas*

in Optimization, Corne, D., Dorigo, M., and Glover, F., Eds., McGraw-Hill, New York, 1999, p. 279.

[48] Dorne, R. and Hao, J. K., A new genetic local search algorithm for graph coloring, in *Parallel Problem Solving From Nature V*, Eiben, A. E., Bäck, Th., Schoenauer, M., and Schwefel, H.-P., Eds., Lecture Notes in Computer Science, Vol. 1498, Springer, Berlin, 1998, p. 745.

[49] Beasley, J. and Chu, P. C., A genetic algorithm for the set covering problem, *Eur. J. Oper. Res.*, 94(2), 393, 1996.

[50] Chu, P. C. and Beasley, J., A genetic algorithm for the generalised assignment problem, *Comput. Oper. Res.*, 24, 17, 1997.

[51] Cotta, C. and Troya, J. M., A hybrid genetic algorithm for the 0-1 multiple knapsack problem, in *Artificial Neural Nets and Genetic Algorithms 3*, Smith, G. D., Steele, N. C., and Albrecht, R. F., Eds., Springer Verlag, Wien, 1998, p. 251.

[52] Kaige, S., Murata, T., and Ishibuchi, H., Performance evaluation of memetic EMO algorithms using dominance relation-based replacement rules on MOO test problems, *IEEE Int. Conf. on Systems, Man and Cybernetics*, 2003, p. 14.

[53] Sörensen, K. and Sevaux, M., MA|PM: Memetic algorithms with population management, *Comput. Oper. Res.*, 33(5), 1214–1225, 2006.

[54] Merz, P. and Freisleben, B., A genetic local search approach to the quadratic assignment problem, *Proc. 7th Int. Conf. on Genetic Algorithms*, 1997, p. 465.

[55] Merz, P. and Freisleben, B., A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem, *Congress on Evolutionary Computation (CEC'99)*, 1999, p. 2063.

[56] Merz, P. and Freisleben, B., Greedy and local search heuristics for the unconstrained binary quadratic programming problem, *J. Heuristics*, 8(2), 197, 2002.

[57] Levine, D., A parallel genetic algorithm for the set partitioning problem, in *Meta-Heuristics: Theory & Applications*, Osman, I. H. and Kelly, J. P., Eds., Kluwer Academic Publishers, Dordrecht, 1996, p. 23.

[58] Mendes, A., França, P., Moscato, P., and Garcia, V., Population studies for the gate matrix layout problem, *IBERAMIA*, Lecture Notes in Computer Science, Vol. 2527, Springer, Berlin, 2002, p. 319.

[59] Mendes A. and Linhares, A., A multiple-population evolutionary approach to gate matrix layout, *Int. J. Syst. Sci.*, 35(1), 13, 2004.

[60] Buriol, L., França, P. M., and Moscato, P., A new memetic algorithm for the asymmetric traveling salesman problem, *J. Heuristics*, 10(5), 483, 2004.

[61] Krasnogor, N. and Smith, J., A memetic algorithm with self-adaptive local search: TSP as a case study, in *Proc. Genetic and Evolutionary Computation Conf. (GECCO-2000)*, Whitley, D. et al., Eds., Morgan Kaufmann, Las Vegas, Nevada, USA, 2000, p. 987.

[62] Blesa, M. J., Moscato, P., and Xhafa, F., A memetic algorithm for the minimum weighted k-cardinality tree subgraph problem, in *Proc. 4th Metaheuristic Int. Conf. (MIC'2001)*, de Sousa, J. P., and Resende, M., Eds., Kluwer Academic Publishers, Boston MA, 2003, pp. 65–90.

[63] Merz, P., A comparison of memetic recombination operators for the traveling salesman problem, *Proc. Genetic and Evolutionary Computation Conf.*, 2002, p. 472.

[64] Rodrigues, A. M. and Soeiro Ferreira, J., Solving the rural postman problem by memetic algorithms, *Proc. 4th Metaheuristic Int. Conf. (MIC'2001)*, 2001, p. 679.

[65] Zou, P., Zhou, Z., Chen, G., and Yao, X., A novel memetic algorithm with random multi-local-search: a case study of tsp, *Congress on Evolutionary Computation*, 2004, p. 2335.

[66] Yeh, W.-C., A memetic algorithm for the min k-cut problem, *Contr. Intell. Syst.*, 28(2), 47, 2000.

[67] Abdinnour, H. S., A hybrid heuristic for the uncapacitated hub location problem, *Eur. J. Oper. Res.*, 106(2–3), 489, 1998.

[68] Hopper, E. and Turton, B., A genetic algorithm for a 2d industrial packing problem, *Comput. Ind. Eng.*, 37(1–2), 375, 1999.

[69] Krzanowski, R. M. and Raper, J., Hybrid genetic algorithm for transmitter location in wireless networks, *Comput. Environ. Urban Syst.*, 23(5), 359, 1999.

[70] Schnecke, V. and Vornberger, O., Hybrid genetic algorithms for constrained placement problems, *IEEE Trans. Evol. Comput.*, 1(4), 266, 1997.

[71] Berger, J., Salois, M., and Begin, R., A hybrid genetic algorithm for the vehicle routing problem with time windows, in *Advances in Artificial Intelligence. Twelfth Biennial Conf. Canadian Society for Computational Studies of Intelligence*, Mercer, R. E. and Neufeld, E., Eds., Springer, Berlin, 1998, p. 114.

[72] Jih, W. R. and Hsu, Y. J., Dynamic vehicle routing using hybrid genetic algorithms, *Proc. Congress on Evolutionary Computation*, 1999, p. 453.

[73] Prins, C. and Bouchenoua, S., A memetic algorithm solving the VRP, the CARP and general routing problems with nodes, edges and arcs, *Advances in Nature-Inspired Computation: The PPSN VII Workshops*, 2002, p. 12.

[74] Hadj-Alouane, A. B., Bean, J. C., and Murty, K. G., A hybrid genetic/optimization algorithm for a task allocation problem, *J. Scheduling*, 2(4), 189–201, 1999.

[75] Klau, G. W., Ljubic, I., Moser, A., Mutzel, P., Neuner, P., Pferschy, U., Raidl, G., and Weiskircher, R., Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem, Lecture Notes in Computer Science, Vol. 3102, Springer, Berlin, 2004, p. 1304.

[76] Altiparmak, F., Dengiz, B., and Smith, A. E., Optimal design of reliable computer networks: a comparison of metaheuristics, *J. Heuristics*, 9(6), 471, 2003.

[77] Quintero, A. and Pierre, S., Sequential and multi-population memetic algorithms for assigning cells to switches in mobile networks, *Comput. Networks*, 43(3), 247, 2003.

[78] Runggeratigul, S., A memetic algorithm to communication network design taking into consideration an existing network, *Proc. 4th Metaheuristic Int. Conf. (MIC'2001)*, 2001, p. 91.

[79] Kersting, S., Raidl, G. R., and Ljubic, I., A memetic algorithm for vertex-biconnectivity augmentation, in *Applications of Evolutionary Computing*, Cagnoni, S. et al., Eds., Lecture Notes in Computer Science, Vol. 2279, Springer, Berlin, 2002, p. 102.

[80] Cotta, C., Scatter search and memetic approaches to the error correcting code problem, in *Evolutionary Computation in Combinatorial Optimization*, Gottlieb, J. and Raidl, G. R., Eds., Lecture Notes in Computer Science, Vol. 3004, Springer, Berlin, 2004, p. 51.

[81] Buriol, L., Resende, M. G. C., Ribeiro, C. C., and Thorup, M., A memetic algorithm for OSPF routing, *Proc. 6th INFORMS Telecommunications Conf.*, 2002, p. 187.

[82] Burke, E. K. and Smith, A. J., A memetic algorithm for the maintenance scheduling problem, *Proc. ICONIP/ANZIIS/ANNES Conf.*, 1997, p. 469.

[83] Cheng, R., Gen, M., and Tsujimura, Y., A tutorial survey of job-shop scheduling problems using genetic algorithms. II. Hybrid genetic search strategies, *Comput. Ind. Eng.*, 37(1–2), 51, 1999.

[84] França, P. M., Gupta, J. N. D., Mendes, A. S., Moscato, P., and Veltink, K. J., Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups, *Comput. Ind. Eng.*, 48(3), 491, 2005.

[85] Ishibuchi, H., Yoshida, T., and Murata, T., Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling, *IEEE Trans. Evol. Comput.*, 7(2), 204, 2003.

[86] Sevaux, M., Jouglet, A., and Oğuz, C., Combining constraint programming and memetic algorithm for the hybrid flowshop scheduling problem, *ORBEL 19th Annual Conf. SOGESCI-BVWB*, Louvain-la-Neuve, Belgium, 2005.

[87] França, P. M., Mendes, A. S., and Moscato, P., Memetic algorithms to minimize tardiness on a single machine with sequence-dependent setup times, *Proc. 5th Int. Conf. Decision Sciences Institute*, 1999, p. 1708.

[88] Maheswaran, R., Ponnambalam, S. G., and Aravindan, C., A meta-heuristic approach to single machine scheduling problems, *Int. J. Adv. Manuf. Technol.*, 25(7–8), 772, 2005.

[89] Mendes, A. S., França, P. M., and Moscato, P., Fitness landscapes for the total tardiness single machine scheduling problem, *Neural Network World, Int. J. Neural Mass-Parallel Comput. Inf. Syst.*, 12(2), 165, 2002.

[90] Mendes, A. S., Muller, F. M., França, P. M., and Moscato, P., Comparing meta-heuristic approaches for parallel machine scheduling problems with sequence-dependent setup times, *Proc. 15th Int. Conf. on CAD/CAM Robotics and Factories of the Future*, Vol. 1, 1999, p. 1.

[91] Ramat, E., Venturini, G., Lente, C., and Slimane, M., Solving the multiple resource constrained project scheduling problem with a hybrid genetic algorithm, *Proc. 7th Int. Conf. on Genetic Algorithms*, 1997, p. 489.

[92] Dellaert, N. and Jeunet, J., Solving large unconstrained multilevel lot-sizing problems using a hybrid genetic algorithm, *Int. J. Prod. Res.*, 38(5), 1083, 2000.

[93] Alkan, A. and Ozcan, E., Memetic algorithms for timetabling, *Congress on Evolutionary Computation*, Canberra, Australia, 2003, p. 1796.

[94] Paechter, B., Cumming, A., Norman, M. G., and Luchian, H., Extensions to a memetic timetabling system, in *The Practice and Theory of Automated Timetabling*, Burke, E. K., and Ross, P., Eds., Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin, 1996, p. 251.

[95] de Causmaecker, P., van den Berghe, G., and Burke, E. K., Using tabu search as a local heuristic in a memetic algorithm for the nurse rostering problem, *Proc. 13th Conf. on Quantitative Methods for Decision Making*, 1999.

[96] Costa, D., An evolutionary tabu search algorithm and the NHL scheduling problem, *INFOR*, 33(3), 161, 1995.

[97] Schonberger, J., Mattfeld, D. C., and Kopfer, H., Memetic algorithm timetabling for non-commercial sport leagues, *Eur. J. Oper. Res.*, 153(1), 102, 2004.

[98] Lim, A., Rodrigues, B., and Zhu, Y., Airport gate scheduling with time windows, *Artif. Intell. Rev.*, 24(1), 5, 2005.

[99] Zhu, Y., Lim, A., and Rodrigues, B., Aircraft and gate scheduling with time windows, *Proc. 15th IEEE Int. Conf. on Tools with Artificial Intelligence*, 2003, p. 189.

[100] Berretta, R. and Rodrigues, L. F., A memetic algorithm for a multistage capacitated lot-sizing problem, *Int. J. Prod. Econ.*, 87(1), 67, 2004.

[101] Torres-Velázquez, R. and Estivill-Castro, V., A memetic algorithm guided by quicksort for the error-correcting graph isomorphism problem, in *Applications of Evolutionary Computing*, Cagnoni, S. et al., Eds., Lecture Notes in Computer Science, Vol. 2279, Springer, Berlin, p. 173.

[102] Bayley, M. J., Jones, G., Willett, P., and Williamson, M. P., Genfold: a genetic algorithm for folding protein structures using NMR restraints, *Protein Sci.*, 7(2), 491, 1998.

[103] Bazzoli, A. and Tettamanzi, A., A memetic algorithm for protein structure prediction in a 3d-lattice hp model, in *EvoWorkshops*, Raidl, G. R. et al., Eds., Lecture Notes in Computer Science, Vol. 3005, Springer, Berlin, 2004, p. 1.

[104] Merz, P., Analysis of gene expression profiles: an application of memetic algorithms to the minimum sum-of-squares clustering problem, *BIOSYSTEMS*, 72(1–2), 99, 2003.

[105] Pacheco J. and Valencia, O., Design of hybrids for the minimum sum-of-squares clustering problem, *Comput. Stat. Data Anal.*, 43(2), 235, 2003.

[106] Speer, N., Spieth, C., and Zell, A., A memetic co-clustering algorithm for gene expression profiles and biological annotation, *Congress on Evolutionary Computation*, 2004, p. 1631.

III

Multiobjective Optimization, Sensitivity Analysis, and Stability

28 Approximation in Multiobjective Problems

Eric Angel
University of Evry Val d'Essonne

Evripidis Bampis
University of Evry Val d'Essonne

Laurent Gourvès
University of Evry Val d'Essonne

28.1 Introduction

In classical combinatorial optimization, the quality of a solution is often measured with a single objective function while, in practice, a decision is rarely made with only one criterion. Then, several conflicting criteria are necessary to measure the quality of a solution. This is the case in a number of areas including transportation, communication, biology, finance, and also computer science [1]. Multiobjective combinatorial optimization (MCO) arises in situations where at least two objective functions are simultaneously taken into account. This framework offers more freedom, and models coming from MCO better fit to real situations than those being from classical combinatorial optimization. Of course, this higher freedom has a price which often turns out to be a higher degree in complexity.

In monocriterion optimization, the notion of optimality is clear while it is not the case when multiple objectives are considered. Conflicts between the competing criteria lead to a frequent situation where no feasible solution meet optimality for all objectives. Instead of a unique and ideal solution, a set of incomparable alternatives *dominating* all the others acts as an optimum. Many years before the emergence of computer science, Vilfredo Pareto gave a characterization of these dominating solutions known today as *Pareto optima* or *Pareto set*. The Pareto set captures the notion of trade-off and its computation is problematic for mainly two reasons:

- it is typically exponential in size; and
- computing one of these *Pareto optima* is often **NP**-hard.

Some problems, known to be polynomially solvable in the monocriterion optimization framework, become **NP**-hard when two (or more) objectives are simultaneously taken into account. Consequently, approximation in multiobjective optimization seems unavoidable.

Since Pareto's pioneering work on optimality, different approaches to tackle a multiobjective problem were proposed. One can divide them into three classes:

- turning a multiobjective problem into a monocriterion one;
- optimize each objective separately; and
- preserve the multidimensional nature of the quality of a solution.

According to this classification of the various attempts, we aim at giving some results that are representative of the MCO area. Before presenting these results, we propose an introduction to MCO which mainly deals with optimality and complexity.

28.2 Multiobjective Combinatorial Optimization Problems

This section first introduces an general definition of an MCO problem. Afterwards, the notion of optimality in multiobjective optimization is addressed. The end of the section is devoted to computational complexity in the MCO area.

28.2.1 General Definition

An MCO problem gathers the following ingredients:

- a finite set $\mathcal{S}$ of feasible solutions;
- a vector $\vec{f} = (f_1, \ldots, f_k)$ of k objective functions; and
- a vector $\vec{g} = (g_1, \ldots, g_k)$ of k goals so that $g_i = min$ (respectively, $g_i = max$) means f_i has to be minimized (respectively, maximized).

The *image* of a solution $s \in \mathcal{S}$ under $\vec{f}$ is a k-dimensional vector $\vec{f}(s)$. The image of $\mathcal{S}$ under $\vec{f}$ is denoted by $\mathcal{I}$. Then, $\mathcal{S}$ and $\mathcal{I}$ respectively define the *decision space* and the *objective space*. The difference between an objective function and a criterion is blurred and then the words *multicriteria* and *multiobjective* are both used to refer to the same framework.

28.2.2 Notions of Optimality

Unless the criteria are not in conflict with each other, it is difficult to define an appropriate notion of optimality and, given this notion, find an optimum.

28.2.2.1 Optimality of Pareto

The optimality of Pareto is based on the notion of *dominance*. Roughly speaking, a solution s dominates another solution s' if it is at least as "good" as s' on every criteria and strictly "better" for at least one criterion. If we suppose that all objective functions of $\vec{f}$ must be minimized, a solution s dominates a solution s', denoted by $s < s'$, if

- $f_i(s) \leq f_i(s')$ for each component f_i of $\vec{f}$; and
- there exists one objective function f_{i^*} of $\vec{f}$ such that $f_{i^*}(s) < f_{i^*}(s')$.

A solution $s \in \mathcal{S}$ is called *Pareto optimal* if there is no $s' \in \mathcal{S}$ such that $s' < s$. The image $\vec{f}(s)$ of a Pareto optimal solution s is called *efficient*. The subset of all Pareto optimal solutions in $\mathcal{S}$, called the *Pareto set*, is denoted by $\mathcal{S}_{Par}$ while $\mathcal{I}_{Eff}$, also called the *Pareto curve*, is its image under $\vec{f}$.

Typically, computing the whole Pareto set is not required and a subset of it having the same image is often sufficient [2].

28.2.2.2 Other Notions of Optimality

Hierarchical Optimality

A ranking between all objective functions is sometimes assumed. A solution s is preferred to s', denoted by $s <_{hier} s'$, iff $f_j(s) < f_j(s')$, where j is the smallest index in the ranking such that $f_i(s) \neq f_i(s')$.

Min Max *Optimality*

A solution is sometimes said to be optimal if it minimizes the largest coordinate of its image. Then, we try to find a solution s such that $\max\{f_1(s), \ldots, f_k(s)\} = \min_{s' \in \mathcal{S}} \max\{f_1(s'), \ldots, f_k(s')\}$.

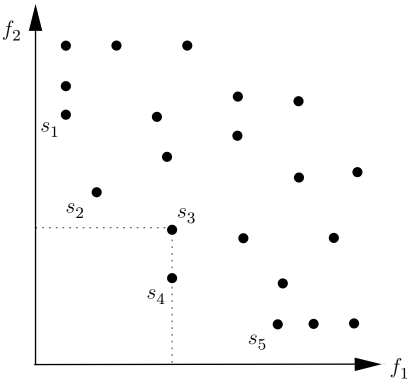


FIGURE 28.1 s_4 dominates s_3. $\{s_1, s_2, s_4, s_5\}$ are Pareto optimal while $\{\vec{f}(s_1), \vec{f}(s_2), \vec{f}(s_4), \vec{f}(s_5)\}$ defines the Pareto curve. s_1 and s_5 are hierarchically optimal. s_3 and s_4 are optimal for the *min max* optimality.

One can remark that the hierarchical optimality is a special case of the Pareto optimality while the *min max* optimality is less demanding than the Pareto optimality since a *min max* optimum is not necessarily Pareto optimal.

Figure 28.1 gives an illustration of the notions given in this subsection.

28.2.3 Hardness

Modeling a problem with more than one objective function offers more freedom but it also brings complications.

28.2.3.1 NP-Completeness

Proving **NP**-completeness for some MCO problems is sometimes easy since it directly follows from **NP**-completeness of an underlying problem with less objectives. For example, this is the case for a multiobjective *traveling salesman problem* (TSP), where the distance between two cities is a vector of distances instead of a scalar. However, some polynomially solvable problems become **NP**-complete when two (or more) criteria are simultaneously taken into account.

As an example, we consider an extension of the well-known *minimal spanning tree problem* called BMST for *bicriteria minimal spanning tree.* We are given a simple graph $G = (V, E)$, where each edge $e \in E$ has a weight w_e and a length l_e. A solution to this problem is a tree t spanning all the vertices of V, having a total weight $W(t) = \sum_{e \in t} w_e$ and a total length $L(t) = \sum_{e \in t} l_e$.

It is well known that the minimal spanning tree problem can be solved in polynomial time [3]. However, we have the following theorem:

Theorem 28.1 (Camerini et al. [4])

BMST *is* **NP**-*complete.*

Proof

Suppose that we have a set A of n integers $\{a_1, \ldots, a_n\}$. Given an integer b, the SUBSET SUM problem asks whether a subset $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = b$ exists. This decision problem is **NP**-complete [5]. Now consider the following decision problem: Given W_0 and L_0, does there exist a spanning tree t that satisfies $W(t) \leq W_0$ and $L(t) \leq L_0$? Set $W_0 = b$, $L_0 = \sum_{i=1}^{n} a_i - b$ and consider the instance G given in Figure 28.2. We only consider spanning trees containing the edge $[v_{n+1}, v_{n+2}]$ since those which do not cannot be preferred. A tree t satisfying $W(t) \leq W_0$ and $L(t) \leq L_0$ yields a subset A' verifying $\sum_{a_i \in A'} a_i = b$ if $a_i \in A' \Leftrightarrow [v_i, v_{n+1}] \in t$. Indeed, $\sum_{e \in t} w_e \leq b \Leftrightarrow \sum_{a_i \in A'} a_i \leq b$ and $\sum_{e \in t} l_e \leq \sum_{i=1}^{n} a_i - b \Leftrightarrow \sum_{a_i \in A'} a_i \geq b$. □

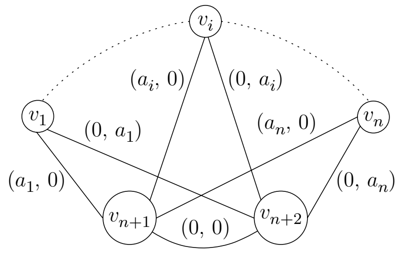


FIGURE 28.2 G has $n+2$ vertices $\{v_1, \ldots, v_{n+2}\}$ and each edge e has a couple (w_e, l_e). Does G admit a spanning tree t such that $W(t) \leq W_0$ and $L(t) \leq L_0$?

Thus, several MCO problems generalizing some polynomially solvable ones are **NP**-complete. For example, this is also the case for SHORTEST PATH or ASSIGNMENT [6]. Sometimes, the complexity relies on the notion of optimality. Finding a hierarchical optimum for the BMST problem can be easily done with a slightly modified version of Kruskal's algorithm where the edges are sorted lexicographically [3]. However, finding a *min max* optimum for the BMST problem is hard because of PARTITION (see the instance in Figure 28.2 and set $W_0 = L_0 = \frac{1}{2}\sum_{i=1}^{n} a_i$).

28.2.3.2 Intractability

Besides the complexity of generating one Pareto optimal solution, computing the Pareto set is problematic since it is often exponential in size and thus, there is no chance to compute it in an efficient manner. In the following, an MCO problem is said *intractable* if $|\mathcal{I}_{Eff}|$ can be exponential in the size of an instance.

We consider an extension of a classical problem called BSP for *bicriteria shortest path.* We are given a graph $G = (V, A)$, where each arc $a \in A$ has a cost c_a and a delay d_a. Given two vertices s and t of the vertex set V, a feasible solution p is a path from s to t whose total cost and total delay are, respectively, $C(p) = \sum_{a \in p} c_a$ and $D(p) = \sum_{a \in p} d_a$.

Theorem 28.2 (**Hansen et al.** [7])

BSP *is intractable.*

Proof

Consider the instance given in Figure 28.3. There are two arcs between two consecutive nodes v_i and v_{i+1}, one with cost–delay vector $(2^i, 0)$ and another with cost–delay vector $(0, 2^i)$. Each path p between s and t satisfies $\sum_{a \in p}(c_a + d_a) = 2^{n+1} - 1$. Moreover, for every positive integer $z \leq 2^{n+1} - 1$ we can build a path p_z such that $C(p_z) = z$. Then, $|\mathcal{I}_{Eff}| = 2^{n+1}$ and BSP is intractable. A similar result on nonoriented graphs can be derived. □

A proof of intractability for BMST can be found in Ref. [8] and the bicriteria ASSIGNMENT problem is addressed in Ref. [9].

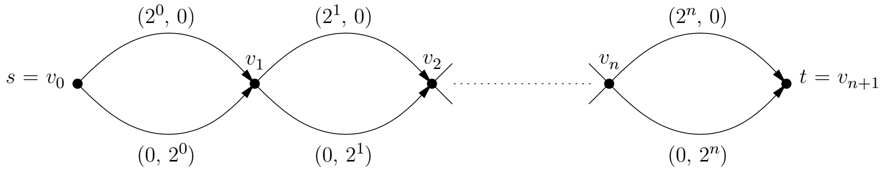


FIGURE 28.3 Each arc a has a cost–delay vector (c_a, d_a). Two different $s-t$ paths in G cannot have the same total cost and total delay.

28.3 Multiobjective Approximation

Three main classes of approaches exist in MCO. Two approaches of the first class are presented in subsections 28.3.1 and 28.3.2. Subsections 28.3.3 and 28.3.4 are, respectively, devoted to the second and third class. Short examples from the literature illustrate the approaches.

28.3.1 Aggregation Approach

One of the most natural ways to address a multiobjective problem is to turn it into a monocriterion one. Instead of dealing with k objective functions, we can aggregate them and get a single function $\tilde{f}$ defined as follows:

$$\tilde{f} = \sum_{i=1}^{k} \lambda_i f_i, \quad \text{where } \lambda_i \in I\!R^+ \text{ and } \sum_{i=1}^{k} \lambda_i = 1$$

Then, considering different values for λ_i allows us to explore different regions of the objective space. This technique consists in searching the objective space with a hyperplane (see Figure 28.4 for an illustration on a biobjective problem).

Interestingly, this *aggregation approach* gives the opportunity to generate some Pareto optimal solutions with an exact monocriterion algorithm. Indeed, any solution s which minimizes $\tilde{f}$, where $\lambda_i > 0$ for all i is Pareto optimal since no other solution s' such that $s' < s$ and $\tilde{f}(s') \geq \tilde{f}(s)$ can exist. By definition, $s' < s$ means $f_i(s') \leq f_i(s)$ for all i and there is a criterion i^* such that $f_{i^*}(s') < f_{i^*}(s)$. Hence, $s' < s$ and $\lambda_i > 0$ imply $\tilde{f}(s') < \tilde{f}(s)$ and contradicts the fact that s minimizes $\tilde{f}$.

The fact that the objective space can be nonconvex implies that only a subset of the Pareto set can be generated with this approach. Then, Pareto optimal solutions are often partitioned into two sets, the *supported* ones which can be computed with the aggregation approach and the *nonsupported* ones (see the encircled dots in Figure 28.4). In early articles addressing MCO, nonsupported solutions were often ignored.

28.3.1.1 A Biobjective Scheduling Problem

We address the problem of scheduling n jobs on a single machine without preemption. Each job $j \in \{1, \ldots, n\}$ has a processing time p_j, a cost c_j, and a weight w_j. We assume that p_j, c_j, and w_j are positive integers. All jobs are available at time $t = 0$ and no precedence constraints are taken into account. Only schedules without idle times are considered and then a feasible solution is a permutation π of the jobs. If a job j is at the ith position in π (i.e., $\pi(i) = j$) then its *completion time* C_j^π is equal to $\sum_{q=1}^{i} p_{\pi(q)}$. A permutation π has a *total cost* $c(\pi) = \sum_{j=1}^{n} c_j C_j^\pi$ and a *total weight* $w(\pi) = \sum_{j=1}^{n} w_j C_j^\pi$. These two possibly conflicting objectives have to be minimized. This biobjective scheduling problem is **NP**-complete [10] and

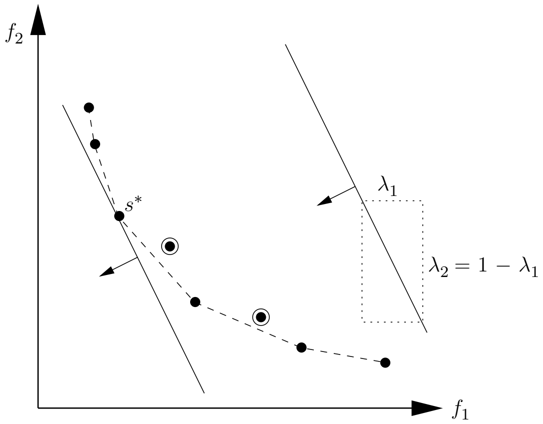


FIGURE 28.4 $\tilde{f} = \lambda_1 f_1 + \lambda_2 f_2$ finds its minimum for s^* which is supported. Nonsupported solutions are encircled.

intractable [11]. We consider a convex combination of the objectives $\tilde{f}_\lambda(\pi) = \lambda c(\pi) + (1-\lambda)w(\pi)$ and we try to compute a set of permutations P such that for any λ between 0 and 1, there is a permutation $\pi \in P$ that minimizes $\tilde{f}_\lambda$. This set is called a *minimal set of supported solutions.*

Each objective can be separately optimized with the rule of Smith [12], since a permutation π that minimizes the total cost (respectively, the total weight) can be computed in time $\mathcal{O}(n \log n)$ if the jobs are sorted by their nondecreasing ratios p_j/c_j (respectively, p_j/w_j). Therefore, an optimal permutation for $\tilde{f}_\lambda$ can be found efficiently if the jobs are sorted by their nondecreasing ratios $r(\lambda, j) = p_j/(\lambda c_j + (1-\lambda)w_j)$. Given a pair of distinct jobs j and j' such that $p_j/c_j \le p_{j'}/c_{j'}$, their ordering in an optimal permutation for $\tilde{f}_\lambda$ does not depend on λ when $p_j/w_j \le p_{j'}/w_{j'}$ (the criteria are not in conflict) but depends on λ when $p_j/w_j > p_{j'}/w_{j'}$ (the criteria are conflicting). Then, one can calculate $\lambda_{jj'}$ such that $r(\lambda_{jj'}, j) = r(\lambda_{jj'}, j')$ for each pair of jobs in conflict and state that the ordering between j and j' in any permutation minimizing $\tilde{f}_\lambda$ depends on the position of λ compared to $\lambda_{jj'}$ [11].

Algorithm (*MSUP*)
returns a minimal set of supported solutions

```
let $\Lambda$ be an ordered list of distinct reals, set $\Lambda = \{0, 1\}$, set $P = \emptyset$
for each pair $j\ j'$ of conflicting jobs
    find $\lambda_{j\,j'}$ such that $0 < \lambda_{j\,j'} < 1$ and $r(\lambda_{j\,j'}, j) = r(\lambda_{j\,j'}, j')$, insert $\lambda_{j\,j'}$ into $\Lambda$
let $\Lambda(i)$ be the $i$th element of $\Lambda$
for $i = 1$ to $|\Lambda| - 1$
    take arbitrarily a $\lambda$ such that $\Lambda(i) < \lambda < \Lambda(i+1)$
    sort the jobs by their nondecreasing ratios $r(\lambda, j)$ and put the permutation in $P$
return $P$
```

Finally, P contains exactly $|\Lambda| - 1$ permutations while $|\Lambda| \le \frac{n(n-1)}{2} + 2$. Thus, *MSUP* runs in $\mathcal{O}(n^3 \log n)$.

28.3.2 Budget Approach

This approach consists in turning a k objective problem into a monocriterion one subject to $k-1$ budget constraints. It is like viewing KNAPSACK as a problem with two objectives: utility and size. More generally (the minimization of each objective is assumed), we are given a $(k-1)$-dimensional vector $(B_2, \ldots, B_k)$ and one tries to minimize $f_1(s)$ such that $s \in \mathcal{S}$, $f_2(s) \le B_2, \ldots, f_k(s) \le B_k$. Let $B_1 = \min_{s \in \mathcal{S}}\{f_1(s) \mid f_i(s) \le B_i, i = 2, \ldots, k\}$; a solution s is called ϵ-approximate if $f_1(s) \le (1+\epsilon)B_1$ and $f_i(s) \le B_i$, $i = 2, \ldots, k$.

28.3.2.1 The Bicriteria Shortest Path Problem on Acyclic Graphs

We consider the BSP problem on acyclic digraphs with a budget approach. Namely, one tries to compute a path p between two nodes s and t such that $D(p) \le D_0$ and $C(p)$ is minimized (D_0 is an entry of the instance). An s–t path with total delay at most D_0 is also called a D_0-path in the following. The material presented in this subsection comes from the articles [13–16].

There is a pseudopolynomial dynamic programming-like procedure to solve the problem. Let $\delta_j(z)$ be the minimal delay for an s–j path of total cost at most z.

Algorithm (*EXACT*(*c*,*d*))
returns a minimal cost D_0-path

```
for all $z \ge 0$, set $\delta_s(z) = 0$
for all $j \ne s$, set $\delta_j(z) = \infty$
for $z = 1, 2, \ldots$
    for all $j \ne s$, set $\delta_j(z) = \min_{ij:\, c_{ij} \le z}\{\delta_j(z-1), \min\{\delta_i(z - c_{ij}) + d_{ij}\}\}$
    if $\delta_t(z) \le D_0$ then output $OPT = z$ and its corresponding path, exit
```

The running time of *EXACT* is $\mathcal{O}(m\ OPT)$, where $m = |A|$ and *OPT* is the minimal cost for a D_0-path. Using a *rounding-and-scaling* technique, one can design a fully polynomial time approximation scheme (FPTAS) for the problem. This technique consists in replacing c (the function that maps a cost to each arc) by a scaled down function $\tilde{c}$ so that *EXACT* $(\tilde{c}, d)$ runs in polynomial time and an optimum for the scaled down instance is an approximation with bounded error of the original optimum.

Suppose that we have two bounds *LB* and *UB* satisfying $LB \leq OPT \leq UB$ and there is a constant $\theta > 1$ such that $UB \leq \theta LB$. Given an ϵ and $n = |V|$, the costs on the original instance are scaled down as follows: $\tilde{c}_a = \lfloor n\, c_a/(\epsilon\, LB)\rfloor$, $\forall a \in A$. Since $\lfloor x \rfloor$ is between $x - 1$ and x, any path p satisfies

$$\sum_{a\in p}\left(\frac{nc_a}{\epsilon\, LB} - 1\right) \leq \sum_{a\in p}\tilde{c}_a \leq \sum_{a\in p}\frac{nc_a}{\epsilon LB}$$

Suppose that p' is an optimal solution for the scaled down instance while p^* is optimal for the original one.

$$\begin{aligned}
\sum_{a\in p'}\tilde{c}_a &\leq \sum_{a\in p^*}\tilde{c}_a\\
\sum_{a\in p'}\left(\frac{n\, c_a}{\epsilon\, LB} - 1\right) &\leq \sum_{a\in p^*}\frac{n\, c_a}{\epsilon\, LB}\\
\sum_{a\in p'}\left(c_a - \frac{\epsilon\, LB}{n}\right) &\leq \sum_{a\in p^*} c_a = OPT\\
C(p') - \epsilon\, LB &\leq OPT\\
C(p') &\leq (1+\epsilon)OPT
\end{aligned}$$

Thus, running *EXACT*$(\tilde{c}, d)$ for $z = 1, 2, \ldots, \lfloor\theta\ n/\epsilon\rfloor$, we get an ϵ-approximate D_0-path in time $\mathcal{O}(nm/\epsilon)$. The remaining part of the result concerns efficient ways to get *LB* and *UB*. Let $c_1 < c_2 < \cdots < c_l$ be the distinct costs of the arcs in A. We clearly have $l \leq m$. Let $A_i = \{a \in A \mid c_a \leq c_i\}$ for $1 \leq i \leq l$, $A_0 = \emptyset$, and $G_i = (V, A_i)$. Therefore, there must be an index j such that G_j admits a D_0-path while G_{j-1} does not. Given this j, one can claim that *OPT* is in the range $[c_j, nc_j]$ since G_{j-1} has no D_0-path means that any D_0-path in G uses at least one arc a such that $c_a \geq c_j$, and *OPT* is upper-bounded by the minimal cost of a D_0-path in G_j which is itself upper-bounded by nc_j. The following algorithm called *INDEX* finds j in $\mathcal{O}(\log\ m)$ steps.

Algorithm (*INDEX*)
returns the minimal j such that G_j admits a D_0-path

```
set $low = 1$ and $high = l$
while $low < high - 1$
    set $j = \lfloor (high + low)/2 \rfloor$
    find an $s - t$ path $p$ in $G_j$ with minimal total delay
    if $p$ exists and $D(p) < D_0$ then $high = j$ else $low = j$
return $high$
```

Thus, *INDEX* helps to get a lower and an upper bound which are within a factor n. Starting with these bounds, one can narrow the range so that $UB/LB \leq 2$. The idea is to iteratively use a procedure which, given $K \in [LB, UB]$ and $\gamma \leq n$, tests if $OPT > K$ or $OPT \leq K(1+\gamma)$:

Algorithm (*TEST*(K, γ))
checks if $OPT > K$ or $OPT \leq K(1+\gamma)$

```
for all $a \in A$, set $\tilde{c}_a = \lfloor (n\tilde{c}_a)/(K\gamma) \rfloor$
for all $z \geq 0$, set $\delta_s(z) = 0$
for all $j \neq s$, set $\delta_j(z) = \infty$
for $z = 1, 2, \ldots, \lfloor n/\gamma \rfloor$
    for all $j \neq s$, set $\delta_j(z) = \min_{ij:\tilde{c}_{ij}\leq z}\{\delta_j(z-1), \min\{\delta_i(z - \tilde{c}_{ij}) + d_{ij}\}\}$
        if $\delta_t(z) \leq D_0$ then output $OPT \leq K(1+\gamma)$ and exit
output $OPT > K$
```

TEST(K, γ) runs in $\mathcal{O}(mn/\gamma)$ time for $\gamma \le n$. Now, we can present the FPTAS for BSP called *ADAPT*.

Algorithm (*ADAPT*(ϵ))
returns an ϵ-approximate D_0-path

```
use $INDEX$ to initialize $UB$ and $LB$ such that $UB \le nLB$
while $UB > 2LB$
    set $\gamma = \sqrt{UB/LB} - 1$ and $K = \sqrt{UB\,LB/(1+\gamma)}$
    if $TEST(K, \gamma)$ outputs $OPT > K$ then set $LB = K$ else set $UB = (1+\gamma)K$
invoke $EXACT(\lfloor nc/(LB\epsilon) \rfloor, d)$ and output its result
```

Theorem 28.3 (Ergun et al. [13])
ADAPT has running time $\mathcal{O}(mn/\epsilon)$.

Proof
Let UB_i, LB_i, K_i, and γ_i be the parameters used in the ith application of *TEST*. The ratio UB_{i+1}/LB_{i+1} is sometimes equal to UB_i/K_i and sometimes equal to $K_i(1+\gamma_i)/LB_i$ but we always have

$$UB_{i+1}/LB_{i+1} = (UB_i/LB_i)^{3/4} \tag{28.1}$$

Let q be the number of applications of *TEST* in *ADAPT*. The total time required to have $UB \le 2LB$ is $\sum_{i=1}^{q} \mathcal{O}(mn/\gamma_i)$ and it therefore suffices to show that $\sum_{i=1}^{q} 1/\gamma_i = \mathcal{O}(1)$. The definition of γ_i implies $1/\gamma_i = 1/(\sqrt{UB_i/LB_i} - 1)$. Since $UB_i > 2LB_i$ for $i \le q$, we have $\sqrt{LB_i/UB_i} \le 1/\gamma_i \le (2+\sqrt{2})\sqrt{LB_i/UB_i}$. Hence, $\sum_{i=1}^{q} 1/\gamma_i = \mathcal{O}\big(\sum_{i=1}^{q} \sqrt{LB_i/UB_i}\big)$. We also have

$$\begin{aligned}\sum_{i=1}^{q} \sqrt{LB_i/UB_i} &= \sum_{j=1}^{q} (LB_q/UB_q)^{(1/2)(4/3)^j} \le \sum_{j=1}^{q} 2^{-(1/2)(3/4)^j} \le 2^{(-1/2)} \sum_{i=j}^{q} (2^{-1/6})^j \\ &\le \frac{2^{-1/2}}{1-2^{-1/6}} \le 6.5\end{aligned}$$

The first equality follows from repeatedly applying Eq. (28.1). The first inequality follows from $UB_q > 2LB_q$. The second inequality holds since $2^{-(1/2)(4/3)^{j+1}} \le (2^{-1/6})2^{-(1/2)(4/3)^j}$ for $j \ge 0$. □

The same approach was successfully used for the BMST problem [17,18].

28.3.3 Simultaneous Approach

No ideal solution meeting optimality for all objectives is likely to exist if the criteria are in conflict. However, the image of this solution, called the *ideal point*, can act as a reference to measure the quality of a compromise solution. The *simultaneous approach* or *ideal point approach* consists in giving a feasible solution whose image approximates the ideal point with a performance guarantee on each objective.

Given a k-criteria optimization problem, the ideal point is a vector $\vec{\mu} = (\mu_1, \ldots, \mu_k)$ such that

$$\mu_i = g_i\{f_i(s) \mid s \in \mathcal{S}\}, \quad i = 1, \ldots, k$$

A solution s is $\vec{\epsilon}$-approximate if for all i between 1 and k we have $f_i(s) \le (1+\epsilon_i)\mu_i$ in case $g_i = min$ or $\mu_i \le (1+\epsilon_i) f_i(s)$ in case $g_i = max$.

28.3.3.1 The Bicriteria MAX-CUT Problem

Given an undirected graph $G = (V, E)$ with nonnegative edge weights w_{ij}, the objective of the classical maximal-cut problem (MAX-CUT) is to find a partition of the vertex set V into two subsets S and $\overline{S}$, such that the sum of the weights of the edges having endpoints in different subsets is maximal. Denoted by $W(S, \overline{S})$, the *total weight* of the cut $(S, \overline{S})$ is $\sum_{i \in S,\ j \in \overline{S}} w_{ij}$. Finding a cut of maximal total weight is **NP**-hard [19].

A biobjective version of this problem is considered. In addition to the weight, the edges have a nonnegative length l_{ij}. The *total length* of a cut, denoted by $L(S, \overline{S})$ is equal to $\sum_{i \in S,\ j \in \overline{S}} l_{ij}$. In the following, the ideal point is denoted by (*OPTW, OPTL*).

We are given an instance of the biobjective problem and two cuts $(S_1, \overline{S_1})$ and $(S_2, \overline{S_2})$ which are, respectively, α-approximate for the total weight (i.e., $OPTW \le (1+\alpha)W(S_1, \overline{S_1})$) and α-approximate for the total length (i.e., $OPTL \le (1+\alpha)L(S_2, \overline{S_2})$). These two cuts are supposed to be computed with a deterministic monocriterion α-approximation algorithm AL. One can build a feasible $(1+2\alpha, 1+2\alpha)$-approximate cut with the following algorithm:

Algorithm (*BIAPPROX*(S_1, S_2))
returns a feasible $(1+2\alpha, 1+2\alpha)$-approximate cut

```
if $L(S_1, \overline{S_1}) \ge L(S_2, \overline{S_2})/2$ then return $(S_1, \overline{S_1})$ and exit
if $W(S_2, \overline{S_2}) \ge W(S_1, \overline{S_1})/2$ then return $(S_2, \overline{S_2})$ and exit
return $\big((S_1 \cap S_2) \cup (\overline{S_1} \cap \overline{S_2}), (\overline{S_1} \cap S_2) \cup (S_1 \cap \overline{S_2})\big)$
```

Theorem 28.4 (Angel et al. [20])

BIAPPROX returns a feasible $(1+2\alpha, 1+2\alpha)$*-approximate cut.*

Proof

The vertex set V can be partitioned into four subsets: $X = S_1 \cap S_2$, $Y = \overline{S_1} \cap S_2$, $Z = S_1 \cap \overline{S_2}$, and $T = \overline{S_1} \cap \overline{S_2}$ (see Figure 28.5). *BIAPPROX* potentially returns the cuts $(S_1, \overline{S_1})$, $(S_2, \overline{S_2})$ or $(X \cup T, Y \cup Z)$. If $L(S_1, \overline{S_1}) \ge L(S_2, \overline{S_2})/2$ then $2(1+\alpha)L(S_1, \overline{S_1}) \ge (1+\alpha)L(S_2, \overline{S_2}) \ge OPTL$. So, $(S_1, \overline{S_1})$ is $(\alpha, 1+2\alpha)$-approximate and thus $(1+2\alpha, 1+2\alpha)$-approximate. If $W(S_2, \overline{S_2}) \ge W(S_1, \overline{S_1})/2$ then $2(1+\alpha)W(S_2, \overline{S_2}) \ge (1+\alpha)W(S_1, \overline{S_1}) \ge OPTW$. So, $(S_2, \overline{S_2})$ is $(1+2\alpha, \alpha)$-approximate and thus $(1+2\alpha, 1+2\alpha)$-approximate. If $L(S_1, \overline{S_1}) < L(S_2, \overline{S_2})/2$ and $W(S_2, \overline{S_2}) < W(S_1, \overline{S_1})/2$ then $W(X, Y) + W(Z, T) \ge W(S_1, \overline{S_1})/2 \ge OPTW/(2(1+\alpha))$ and $L(X, Z) + L(Y, T) \ge L(S_2, \overline{S_2})/2 \ge OPTL/(2(1+\alpha))$. As a consequence $(X \cup T, Y \cup Z)$ is $(1+2\alpha, 1+2\alpha)$-approximate. □

Suppose that AL is an exact (0-approximation) algorithm, then *BIAPPROX* provides a constructive proof on the existence of a feasible (1, 1)-approximate cut. Now suppose that AL is the derandomized version of the 0.13821-approximation algorithm of Goemans and Williamson [21] ($0.13821 = 0.87856^{-1} - 1$), *BIAPPROX* returns a feasible (1.276, 1.276)-approximate cut in polynomial time. In contrast to these positive results, one can remark that no (ϵ_1, ϵ_2)-approximation algorithm can be designed for the biobjective MAX CUT when ϵ_1 and ϵ_2 are both strictly inferior to 1. Indeed, the instance depicted in Figure 28.6 shows that no such cut exists.

The same approach was successfully used for a scheduling problem [22].

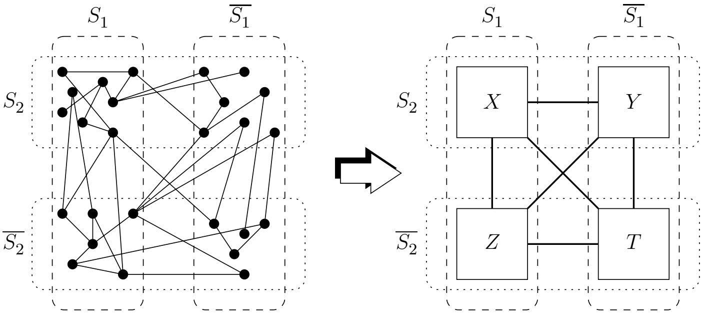


FIGURE 28.5 S_1, $\overline{S_1}$, S_2, and $\overline{S_2}$ define a partition of V into four sets called X, Y, Z, and T.

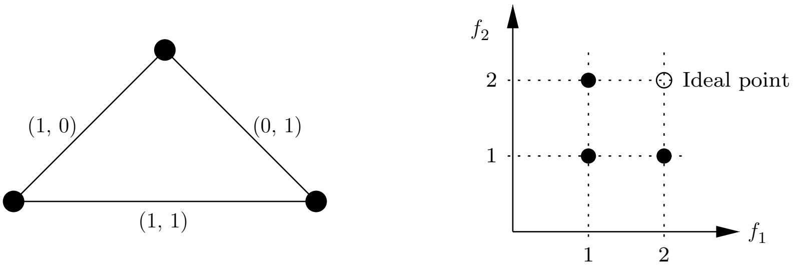


FIGURE 28.6 Each edge e of this graph has a couple (w_e, l_e). The ideal point is (2, 2) while no feasible cut $(S, \overline{S})$ such that $W(S, \overline{S}) > 1$ and $L(S, \overline{S}) > 1$ exists.

28.3.4 Pareto Set Approach

Providing the whole set of Pareto optimal solutions or a subset having the same image under $\vec{f}$ is the best answer one could give to a multiobjective problem. Despite the hardness of computing such a set, one can try to generate in polynomial time an approximation of it. Hence, a succinct set of incomparable solutions which approximately dominate all the others is of great interest.

Given a vector $\vec{\epsilon} = (\epsilon_1, \ldots, \epsilon_k)$, we say that s $\vec{\epsilon}$-dominates s' if $f_i(s) \leq (1 + \epsilon_i) f_i(s')$, $i = 1, \ldots, k$ (the minimization of every objective function is assumed). Hence, an *$\vec{\epsilon}$-approximate Pareto set* $\mathcal{S}_{\vec{\epsilon}}$ is a set of feasible solutions such that for every solution $s' \in \mathcal{S}$ there exists an $s \in \mathcal{S}_{\vec{\epsilon}}$ which $\vec{\epsilon}$-dominates s'. The image of $\mathcal{S}_{\vec{\epsilon}}$ under $\vec{f}$ is called an *$\vec{\epsilon}$-approximate Pareto curve.*

Theorem 28.5 (Papadimitriou and Yannakakis [2])

For any multiobjective optimization problem and any ϵ there is an $\vec{\epsilon}$-approximate Pareto set consisting of a number of solutions that is polynomial in the size of the instance and $1/\epsilon$ but exponential in the number of criteria.

Proof

Given an instance x of a k-criteria problem and a solution $s \in \mathcal{S}$, we assume that $f_i(s)$ is between $2^{-p(|x|)}$ and $2^{p(|x|)}$ for some polynomial p. Consider the objective space and subdivide it into hyperrectangles, such that, in each dimension, the ratio of the larger to the smaller coordinate is $1 + \epsilon$ (see Figure 28.7). There are $\mathcal{O}\left(\frac{(2p(|x|))^k}{\epsilon^k}\right)$ such subdivisions. Define $\mathcal{S}_{\vec{\epsilon}}$ by choosing one solution of $\mathcal{S}_{Par}$ in each hyperrectangle that contains such a solution. □

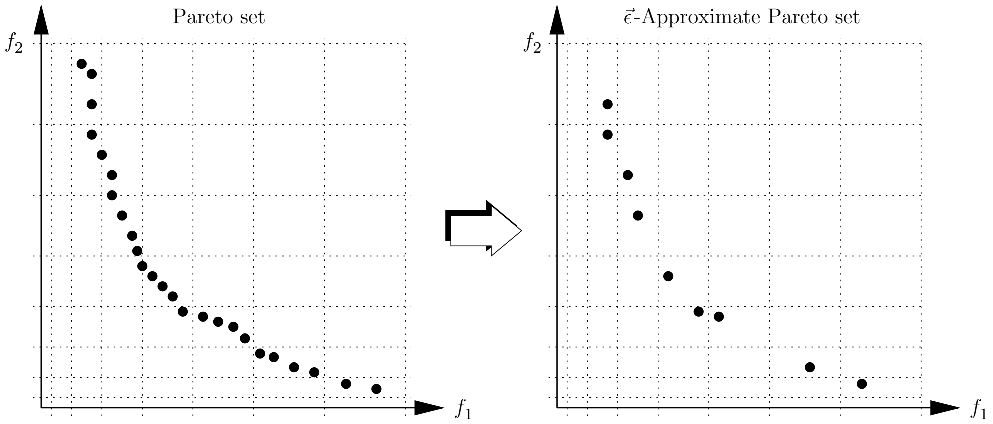


FIGURE 28.7 The geometric grid ensures that the image of two solutions in the same box are within a factor $1 + \epsilon$ on every coordinate. Keeping one solution per box leads to an $\vec{\epsilon}$-approximate Pareto set.

28.3.4.1 An Approximate Pareto Set with a Simultaneous Approach

Let s^* be a Pareto optimal solution for a multiobjective problem. Given $\vec{\mu}$ as the ideal point, we have $f_i(s^*) \geq \mu_i$ when $g_i = min$ and $f_i(s^*) \leq \mu_i$ when $g_i = max$. Let s be an $\vec{\epsilon}$-approximate solution with respect to the simultaneous approach. We have $f_i(s) \leq (1+\epsilon_i)\mu_i$ when $g_i = min$ and $\mu_i \leq (1+\epsilon_i) f_i(s)$ when $g_i = max$. Hence, s $\vec{\epsilon}$-dominates s^* since $f_i(s) \leq (1+\epsilon_i)\mu_i \leq (1+\epsilon_i) f_i(s^*)$ when $g_i = min$ and $f_i(s^*) \leq \mu_i \leq (1+\epsilon_i) f_i(s^*)$ when $g_i = max$. Therefore, $\{s\}$ constitutes an $\vec{\epsilon}$-approximate Pareto set.

28.3.4.2 An Approximate Pareto Set with a Budget Approach

We first address a problem with two objectives f_1 and f_2 in which both have to be minimized. We make the assumption that every solution $s \in \mathcal{S}$ satisfies $LB \leq f_2(s) \leq LB(1+\epsilon)^Q$, where $LB > 0$, $\epsilon > 0$, and Q is a positive integer. Now suppose that we have an ϵ'-approximation algorithm AL with respect to the budget approach. Namely, $AL(B_2)$ returns a solution s such that $f_1(s) \leq (1+\epsilon')B_1$ and $f_2(s) \leq B_2$, where $B_1 = \min_{s' \in \mathcal{S}}\{f_1(s') \mid f_2(s') \leq B_2\}$.

Algorithm ($MULTIBUDGET(LB, \epsilon, AL)$)
returns a (ϵ', ϵ)-approximate Pareto set

```
1: set $P = \emptyset$
2: for $q = 1, \ldots, Q$
3:     set $P = P \cup \{AL(LB(1+\epsilon)^q)\}$
4: output $P$
```

Theorem 28.6

MULTIBUDGET returns an (ϵ', ϵ)*-approximate Pareto set.*

Proof

Take any Pareto optimal solution $s^* \in \mathcal{S}_{Par}$. By the assumption, we know that there exists a q^* such that $1 \leq q^* \leq Q$ and $LB(1+\epsilon)^{q^*-1} \leq f_2(s^*) \leq LB(1+\epsilon)^{q^*}$. Let s' be such that $f_1(s') = \min_{s \in \mathcal{S}}\{f_1(s) \mid f_2(s) \leq LB(1+\epsilon)^{q^*}\}$. We have $f_1(s') \leq f_1(s^*)$. Let $s \in P$ be the solution returned by $AL(LB(1+\epsilon)^{q^*})$. We have $f_1(s) \leq (1+\epsilon') f_1(s')$ and $f_2(s) \leq LB(1+\epsilon)^{q^*}$. Thus, we get $f_1(s) \leq (1+\epsilon') f_1(s^*)$ and $f_2(s) \leq (1+\epsilon) f_2(s^*)$ meaning that s (ϵ', ϵ)-dominates s^*. □

Therefore, an algorithm which outputs an $\vec{\epsilon}$-approximate Pareto set for the BSP problem can be designed if we combine *MULTIBUDGET* and *ADAPT*. This approach acts as a general technique which can be directly adapted for problems with more than two criteria. The minimization of each criterion is assumed in the following theorem:

Theorem 28.7 (Papadimitriou and Yannakakis [2]**)**

There is an algorithm for constructing an $\vec{\epsilon}$-approximate Pareto set, polynomial in the size of the instance and $1/\epsilon$, if the following problem (called GAP*) can be so solved: Given the instance and a k-vector $(B_1, \ldots, B_k)$, either return a solution s with $f_i(s) \leq (1+\epsilon)B_i$ for all i, or answer that there is no solution s′ with $f_i(s') \leq B_i$.*

Proof

Define $\epsilon' = \sqrt{1+\epsilon} - 1 \approx \epsilon/2$ and subdivide the objective space as in the proof of Theorem 28.5, using ϵ'. Call the GAP problem for each corner (see Figure 28.8 for an illustration). Keep (an undominated subset of) all solutions returned and then we get an $\vec{\epsilon}$-approximate Pareto set. □

This approach was successfully used for a scheduling problem [23]. The following subsection addresses a problem where the general technique cannot be used.

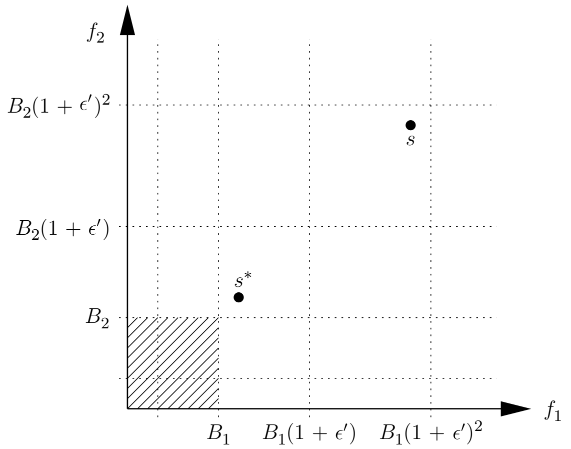


FIGURE 28.8 Suppose that s^* is Pareto optimal. Call GAP for (B_1, B_2) and suppose that no solution s' satisfies $f_i(s') \leq B_i$. Call GAP for $(B_1(1+\epsilon'), B_2(1+\epsilon'))$ and suppose that it returns a solution s with $f_i(s) \leq B_i(1+\epsilon')^2$. Then, we get $f_i(s) \leq (1+\epsilon')^2 f_i(s^*) = (1+\epsilon) f_i(s^*)$.

28.3.4.3 The Bicriteria TSP (1, 2)

Given a complete graph $G = (V, E)$ and a function d that maps a nonnegative distance to each edge $e \in E$, the TSP consists in finding a tour (a Hamiltonian cycle) whose total distance is minimum. Papadimitriou and Yannakakis [24] addressed a restricted, but still **NP**-hard, version of the TSP where distances can be one or two (denoted by TSP(1, 2) in the following). They proposed a polynomial-time 1/6-approximation algorithm, while Engebretsen and Karpinski [25] provided a lower bound of 1/740 on this ratio (for polynomial-time approximation).

We consider a bicriteria version of the TSP(1, 2) where all distance vectors belong to $\{1, 2\}^2$. A tour t has total distance vector $\vec{D}(t)$ such that $n \leq D_k(t) \leq 2n$, $n = |V|$, and $1 \leq k \leq 2$. For any instance of this biobjective problem, one can compute in polynomial time a set P' of two tours such that the image of P' under $\vec{D}$ is a $(\frac{1}{2}, \frac{1}{2})$-approximation of $\mathcal{I}_{Eff}$. Each of these two tours is computed with an extended version of the old *nearest neighbor* heuristic. Starting from a single node, this greedy procedure consists in iteratively adding the "lightest" edge connecting the last inserted node with a noninserted one. A tour is simply built by linking the extremities of the path that was computed. If distances are scalars then the word "lightest" means "minimal distance." In case distances are vectors then a definition of "lightness" must be given since some distance vectors cannot be objectively compared.

Suppose that we are given a permutation σ of the criteria. One can derive a lexicographic order between the distance vectors as follows: According to σ, $\vec{d}(e)$ is lighter than $\vec{d}(e')$, denoted by $\vec{d}(e) <_\sigma \vec{d}(e')$, if $d_{\sigma(j)}(e) < d_{\sigma(j)}(e')$ and j is the smallest index satisfying $d_{\sigma(j)}(e) \neq d_{\sigma(j)}(e')$.

The following algorithm takes a permutation σ as an entry and returns an approximate tour t_σ (t_σ is viewed as a permutation of the nodes so that $t_\sigma(i)$ denotes the ith visited node):

Algorithm (*BINN*(σ))
returns an approximate tour t_σ

```
choose arbitrarily one node $v \in V$
set $t_\sigma(1) = v$ and $VISITED = \{v\}$
for $i = 1, \ldots, n-1$
    find $v' \in V \setminus VISITED$ the nearest neighbor of $t_\sigma(i)$
    set $t_\sigma(i+1) = v'$ and $VISITED = VISITED \cup \{v'\}$
return $t_\sigma$
```

Theorem 28.8 **(Angel et al. [26])**

BINN(σ) returns a tour t_σ which $(\frac{1}{2}, \frac{1}{2})$-dominates any tour t' satisfying $D_{\sigma(1)}(t') \le D_{\sigma(2)}(t')$.

Proof

Given $\vec{a} \in \{1,2\}^2$, let $x_{\vec{a}}$ (respectively, $x'_{\vec{a}}$) be the number of edges in t_σ (respectively, t') having a distance vector equal to $\vec{a}$. For $k = 1$ or $k = 2$, we have $D_k(t_\sigma) = \sum_{\vec{a}:a_k=1} x_{\vec{a}} + 2\sum_{\vec{a}:a_k=2} x_{\vec{a}}$ and $D_k(t') = \sum_{\vec{a}:a_k=1} x'_{\vec{a}} + 2\sum_{\vec{a}:a_k=2} x'_{\vec{a}}$. Since t_σ and t' are two tours with n edges, we have

$$n = \sum_{\vec{a}:a_k=1} x_{\vec{a}} + \sum_{\vec{a}:a_k=2} x_{\vec{a}} = \sum_{\vec{a}:a_k=1} x'_{\vec{a}} + \sum_{\vec{a}:a_k=2} x'_{\vec{a}}, \quad k = 1, 2$$

Thus, $D_k(t_\sigma) = 2n - \sum_{\vec{a}:a_k=1} x_{\vec{a}}$ and $D_k(t') = 2n - \sum_{\vec{a}:a_k=1} x'_{\vec{a}}$. The hypothesis $D_{\sigma(1)}(t') \le D_{\sigma(2)}(t')$ gives

$$\sum_{\vec{a}:a_{\sigma(1)}=1} x'_{\vec{a}} + 2\sum_{\vec{a}:a_{\sigma(1)}=2} x'_{\vec{a}} \le \sum_{\vec{a}:a_{\sigma(2)}=1} x'_{\vec{a}} + 2\sum_{\vec{a}:a_{\sigma(2)}=2} x'_{\vec{a}} \Leftrightarrow \sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=1} x'_{\vec{a}} \le \sum_{\vec{a}:a_{\sigma(1)}=1\wedge a_{\sigma(2)}=2} x'_{\vec{a}} \tag{28.2}$$

Let Ω be a subset of $\{1,2\}^2$ such that no couple of vectors $\vec{a}, \vec{b}$ satisfies $\vec{a} \in \{1,2\}^2\backslash\Omega$, $\vec{b} \in \Omega$ and $\vec{a} \prec_\sigma \vec{b}$. For example, Ω can be $\{(1,1),(1,2)\}$ or $\{(1,1)\}$ if σ is the identity permutation. Let X_Ω (respectively, X'_Ω) be the set of edges in t_σ (respectively, t') which have a distance vector in Ω. Let $map : X'_\Omega \to X_\Omega$ be a function such that $map(e) = e$ if $e \in X'_\Omega \cap X_\Omega$; otherwise $map(e) = e'$, where $e = [t_{\sigma(i)}, t_{\sigma(j)}]$, $e' = [t_{\sigma(i)}, t_{\sigma(i+1)}]$, and $1 \le i < j \le n$. Remark that e' must belong to X_Ω because $t_{\sigma(i+1)}$ is the nearest neighbor of $t_{\sigma(i)}$, i.e., $\vec{d}(e) \not\prec_\sigma \vec{d}(e')$ and $\vec{d}(e') \in \Omega$. Since in a tour, exactly two edges are incident to a vertex, we have $2|X_\Omega| \ge |X'_\Omega|$. Then, taking $\Omega = \{\vec{a} \mid a_{\sigma(1)} = 1\}$ and $\Omega = \{\vec{a} \mid a_{\sigma(1)} = a_{\sigma(2)} = 1\}$, we get

$$2\sum_{\vec{a}:a_{\sigma(1)}=1} x_{\vec{a}} \ge \sum_{\vec{a}:a_{\sigma(1)}=1} x'_{\vec{a}} \tag{28.3}$$

$$2\sum_{\vec{a}:a_{\sigma(1)}=a_{\sigma(2)}=1} x_{\vec{a}} \ge \sum_{\vec{a}:a_{\sigma(1)}=a_{\sigma(2)}=1} x'_{\vec{a}} \tag{28.4}$$

Proving that t_σ $(\frac{1}{2}, \frac{1}{2})$-dominates t' is equivalent to $D_{\sigma(k)}(t_\sigma) \le (1 + \frac{1}{2})D_{\sigma(k)}(t')$ for $k = 1, 2$.

$$\begin{aligned} D_{\sigma(k)}(t_\sigma) \le \left(1 + \frac{1}{2}\right) D_{\sigma(k)}(t') &\Leftrightarrow 2n - \sum_{\vec{a}:a_{\sigma(k)}=1} x_{\vec{a}} \le \frac{3}{2}\Big(2n - \sum_{\vec{a}:a_{\sigma(k)}=1} x'_{\vec{a}}\Big) \\ &\Leftrightarrow -2\sum_{\vec{a}:a_{\sigma(k)}=1} x_{\vec{a}} \le 2n - 3\sum_{\vec{a}:a_{\sigma(k)}=1} x'_{\vec{a}} \\ &\Leftrightarrow -2\sum_{\vec{a}:a_{\sigma(k)}=1} x_{\vec{a}} \le 2\sum_{\vec{a}:a_{\sigma(k)}=2} x'_{\vec{a}} - \sum_{\vec{a}:a_{\sigma(k)}=1} x'_{\vec{a}} \qquad (28.5) \end{aligned}$$

Since $x'_{\vec{a}} \ge 0$ for any vector $\vec{a} \in \{1,2\}^2$, inequality (28.5) when $k = 1$ follows from Eq. (28.3) and $0 \le 2\sum_{\vec{a}:a_{\sigma(1)}=2} x'_{\vec{a}}$.

Starting from Eq. (28.2) we get

$$2\sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=1} x'_{\vec{a}} \le 2\sum_{\vec{a}:a_{\sigma(1)}=1\wedge a_{\sigma(2)}=2} x'_{\vec{a}}$$

$$2\sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=1} x'_{\vec{a}} + 2\sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=2} x'_{\vec{a}} \le 2\sum_{\vec{a}:a_{\sigma(2)}=2} x'_{\vec{a}}$$

$$2\sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=1} x'_{\vec{a}} + 2\sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=2} x'_{\vec{a}} - \sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=1} x'_{\vec{a}} \le 2\sum_{\vec{a}:a_{\sigma(2)}=2} x'_{\vec{a}} - \sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=1} x'_{\vec{a}}$$

$$\sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=1} x'_{\vec{a}} + 2\sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=2} x'_{\vec{a}} \le 2\sum_{\vec{a}:a_{\sigma(2)}=2} x'_{\vec{a}} - \sum_{\vec{a}:a_{\sigma(1)}=2\wedge a_{\sigma(2)}=1} x'_{\vec{a}} \tag{28.6}$$

Since $x'_{\vec{a}} \geq 0$ and $x_{\vec{a}} \geq 0$ for any vector $\vec{a} \in \{1, 2\}^2$, we have

$$-2 \sum_{\vec{a}:a_{\sigma(1)}=2 \wedge a_{\sigma(2)}=1} x_{\vec{a}} \leq \sum_{\vec{a}:a_{\sigma(1)}=2 \wedge a_{\sigma(2)}=1} x'_{\vec{a}} + 2 \sum_{\vec{a}:a_{\sigma(1)}=2 \wedge a_{\sigma(2)}=2} x'_{\vec{a}}$$

This inequality and Eq. (28.6) give

$$-2 \sum_{\vec{a}:a_{\sigma(1)}=2 \wedge a_{\sigma(2)}=1} x_{\vec{a}} \leq 2 \sum_{\vec{a}:a_{\sigma(2)}=2} x'_{\vec{a}} - \sum_{\vec{a}:a_{\sigma(1)}=2 \wedge a_{\sigma(2)}=1} x'_{\vec{a}} \tag{28.7}$$

Finally, one can add Eq. (28.4) to Eq. (28.7) and get inequality (28.5) when $k = 2$. □

Running *BINN* with the two possible permutations σ, one can build a set P' of two tours which $(\frac{1}{2}, \frac{1}{2})$-dominates any feasible tour. Hence, P' is a $(\frac{1}{2}, \frac{1}{2})$-approximate Pareto set for the bicriteria TSP(1, 2).

28.4 Conclusion

Practical situations without any clearly established optimum are often met. Indeed, considering several conflicting criteria is sometimes necessary or simply relevant. Instead of a well-identified optimum, one faces a set of trade-off solutions whose computation is highly problematic. Thus, approximation seems reasonable in multiobjective optimization.

There are various approaches or techniques for the study of a multiobjective problem:

- aggregate the objective functions and get a single criterion which has to be optimized;
- optimize only one objective function while the others are turned into constraints;
- get a trade-off by combining solutions which are good for each separate objective function; and
- compute a set of solutions which approximately dominates all the others.

References

[1] Ehrgott, M. and Gandibleux, X., A survey and annotated bibliography of multiobjective combinatorial optimization, *OR Spektrum*, 22, 425, 2000.

[2] Papadimitriou, C. H. and Yannakakis, M., On the approximability of trade-offs and optimal access of web sources, *Proc. of FOCS,* 2000, p. 86.

[3] Kruskal, J. B., On the shortest spanning subtree of a graph and the travelling salesman problem, *Proc. of AMS*, Vol. 7, 1956, p. 48.

[4] Camerini, P. M., Galbiati, G., and Maffioli, F., The complexity of multi-constrained spanning tree problems, in *Theory of Algorithms, Colloquium PECS*, 1984, János Bolyai Mathematical Society, Budapest, p. 53.

[5] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman, San Francisco, CA, 1979.

[6] Serafini, P., Some considerations about computational complexity for multi-objective combinatorial problems, in *Recent Advances and Historical Development of Vector Optimization*, Jahn, J. and Krabs, W., Eds., Lecture Notes in Economics and Mathematical Systems, Vol. 294, Springer, Berlin, 1986, p. 222.

[7] Hansen, P., Bicriterion path problems, in *Multiple Criteria Decision Making Theory and Application*, Fandel, G. and Gal, T., Eds., Lecture Notes in Economics and Mathematical Systems, Vol. 177, Springer, Berlin, 1979, p. 109.

[8] Hamacher, H. W. and Ruhe, G., On spanning tree problems with multiple objectives, *Ann. Oper. Res.*, 52, 209, 1994.

[9] Ehrgott, M., *Muticriteria Optimization*, Lecture Notes in Economics and Mathematical Systems, Vol. 491, Springer, Berlin, 2000.

[10] Hoogeveen, H., Single-Machine Bicriteria Scheduling, Ph.D. thesis, Eindhoven University of Technology, 1992.

[11] Angel, E., Bampis, E., and Gourvès, L., Approximation results for a bicriteria job scheduling problem on a single machine without preemption, *Inf. Process. Lett.*, 94, 19, 2005.

[12] Smith, W. E., Various optimizers for single-stage production, *Naval Res. Logistics Quart.*, 3, 59, 1956.

[13] Ergun, F., Sinha, R., and Zhang, L., An improved FPTAS for restricted shortest path, *Inf. Process. Lett.*, 83, 287, 2002.

[14] Hassin, R., Approximation schemes for the restricted shortest path problems, *Math. Oper. Res.*, 17, 36, 1992.

[15] Lorenz, D. H. and Raz, D., A simple efficient approximation scheme for the restricted shortest path problem, *Oper. Res. Lett.*, 28, 213, 2001.

[16] Warburton, A., Approximation of Pareto optima in multiple objective, shortest path problems, *Oper. Res.*, 35, 70, 1987.

[17] Hong, S.-P., Chung, S.-J., and Park, B. H., A fully polynomial bicriteria approximation scheme for the constrained spanning tree problem, *Oper. Res. Lett.*, 32, 233, 2004.

[18] Ravi, R. and Goemans, M. X., The constrained minimum spanning tree problem (extended abstract), *Proc. of SWAT*, Lecture Notes in Computer Science, Vol. 1097, 1996, p. 66.

[19] Karp, R. M., Reducibility among Combinatorial problems, in *Complexity of Computer Computations*, Miller, R. and Tatcher, J., Eds., Plenum Press, New York, 1972, p. 85.

[20] Angel, E., Bampis, E., and Gourvès, L., Approximation algorithms for the bi-criteria weighted MAX-CUT problem, *Proc. of WG,* Lecture Notes in Computer Science, 2005.

[21] Goemans, M. X. and Williamson, D. P., 878-approximation algorithms for MAX CUT and MAX 2SAT, *Proc. of STOC*, 1994, p. 422.

[22] Stein, C. and Wein, J., On the existence of schedules that are near-optimal for both makespan and total weighted completion time, *Oper. Res. Lett.*, 21, 115, 1997.

[23] Angel, E., Bampis, E., and Kononov, A., On the approximate tradeoff for bicriteria batching and parallel machine scheduling problems, *Theor. Comput. Sci.*, 306, 319, 2003.

[24] Papadimitriou, C. H. and Yannakakis, M., The traveling salesman problem with distances one and two, *Math. Oper. Res.*, 18, 1, 1993.

[25] Engebretsen, L. and Karpinski, M., Approximation hardness of TSP with bounded metrics, *Proc. of ICALP*, Lecture Notes in Computer Science, Vol. 2076, 2001, p. 201.

[26] Angel, E., Bampis, E., and Gourvès, L., (Non)-approximability for the multi-criteria TSP(1, 2), *Proc. of FCT*, Lecture Notes in Computer Science, Vol. 3623, 2005, p. 318.

29

Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: A Review

Luís Paquete
University of the Algarve

Thomas Stützle
Free University of Brussels

29.1 Introduction

Multiobjective Combinatorial Optimization Problems (MCOPs) are combinatorial problems that involve the optimization of several, typically conflicting objectives. An MCOP arises, for example, when planning a holiday trip from a city A to some city B. Besides the minimization of the overall distance between the two sites, one may also be interested in minimizing the cost, the overall travel time, etc. An unequivocal solution to such problems is the one which is *optimal* with respect to all objectives. But is there such a solution? The shortest tour is not necessarily the fastest nor the fastest needs to be the cheapest one—just consider tolled highways.

Which is then the optimal solution to such a multiobjective problem? It depends on the notion of *optimality*. In this chapter, we focus on the notion of Pareto optimality, which arises when the decision maker is not able to express his preferences a priori, simply because he is not present in the process or not able to give an a priori formula or ranking of the objectives. In this case, one is interested in obtaining a set of solutions that represents the *optimal trade-off* between the objectives, i.e., solutions which are not worse than any other and strictly better in at least one of the objectives.

The set of available algorithms for computing high-quality approximations to the Pareto optimal set has grown enormously over the recent years as witnessed by a large number of papers at international conferences and workshops [1–4], special issues of scientific journals [5–7] and numerous regular papers at multiple criteria decision making as well as at algorithms conferences. The majority of these approaches are based on stochastic local search (SLS) algorithms (see Chapter 19 for more details on SLS), a trend that reflects the enormous success of these algorithms for single-objective problems.

Here, we review the main developments in the application of SLS algorithms to MCOPs. Stochastic local search techniques range from simple constructive algorithms and iterative improvement algorithms to general algorithm frameworks that can be adapted to a specific problem under consideration. These latter general-purpose SLS methods (also often called metaheuristics) include simulated annealing, tabu search, evolutionary algorithms, ant colony optimization (ACO), and many others. While these techniques can result in rather complex algorithms already for single-objective problems, when applied to MCOPs they become even more complicated because they need to return a set of solutions instead of a single one.

For tackling MCOPs with SLS algorithms, two fundamentally different approaches can be distinguished. The first is to base the search on the component-wise ordering of the objective value vectors of solutions (or some *ranking* derived from these orderings). We will say that SLS algorithms that mainly focus on this approach follow the *component-wise acceptance criterion* (CWAC) *search model.* The second approach is based on the usage of parameterized scalarization methods by aggregating the objectives; the SLS algorithms following such lines use the *scalarized acceptance criterion* (SAC) *search model.* These two choices somehow define the two main *schools* for the design of SLS algorithms for MCOPs. Different choices for the remaining components of an SLS algorithm for MCOPs crucially depend on the choice taken for the search model.

In the following sections, we review available SLS algorithms in dependence of these two choices, which also makes this review different from several earlier ones [8–13]. In addition, some proposals combine these two search models. We will review these latter *hybrid* approaches separately from the others. Despite the view taken here, it is inevitable to further discuss the proposed algorithms in dependence of the analogy to known SLS methods for the single-objective problems. However, since we are more interested in the differences between the main search strategies, we do not consider specific problem-dependent implementation choices in detail and also try to avoid the highly specialized jargon found in the context of various SLS methods.

29.2 Basics

The main goal of solving MCOPs in terms of Pareto optimality is to find solutions which are not worse than any other solution and strictly better in at least one of the objectives. Let Q be the number of objectives and $\mathcal{S}$ be the set of all candidate solutions; then the objective function for a solution $s \in \mathcal{S}$ to MCOPs can be defined as a mapping $\vec{f} : s \mapsto \mathbb{R}^Q$. The following orders hold for objective function vectors in $\mathbb{R}^Q$. Let $\vec{u}$ and $\vec{v}$ be vectors in $\mathbb{R}^Q$; we define the (i) *weak component-wise order* as $\vec{u} \leq \vec{v}$, *i.e.*, $u_i \leq v_i,\ i = 1, \ldots, Q$; and (ii) the *component-wise order* as $\vec{u} \prec \vec{v}$, *i.e.*, $\vec{u} \neq \vec{v}$ and $u_i \leq v_i, i = 1, \ldots, Q$. In the context of optimization, we denote the relation between objective function value vectors of two feasible solutions s and s' as follows: (i) if $\vec{f}(s) \prec \vec{f}(s')$, we say that $\vec{f}(s)$ *dominates* $\vec{f}(s')$; and (ii) if $\vec{f}(s) \leq \vec{f}(s')$, then $\vec{f}(s)$ *weakly dominates* $\vec{f}(s')$. In addition, we say that $\vec{f}(s)$ and $\vec{f}(s')$ are *nondominated* if $\vec{f}(s) \nprec \vec{f}(s')$ and $\vec{f}(s') \nprec \vec{f}(s)$, and they are *nonweakly dominated* if $\vec{f}(s) \nleq \vec{f}(s')$ and $\vec{f}(s') \nleq \vec{f}(s)$. Note that the latter implies that $\vec{f}(s) \neq \vec{f}(s')$. For simplification purposes, we shall use the same notation among solutions when the above relations hold between their objective function value vectors.

Since the notion of optimal solution clearly differs from the single-objective counterpart, we need to define the notion of a *Pareto global optimum solution* and a *Pareto global optimum set*: A solution $s \in \mathcal{S}$ is a Pareto global optimum if and only if there is no $s' \in \mathcal{S}$ such that $\vec{f}(s') \prec \vec{f}(s)$; we say that $\mathcal{S}' \subseteq \mathcal{S}$ is a Pareto global optimum set if and only if it contains *only* and *all* Pareto global optimum solutions. We call the image of the Pareto global optimum set in the objective space *the efficient set*. In most cases, solving an MCOP in terms of Pareto optimality would correspond to finding solutions that are representative of the efficient set.

When solving an MCOP in terms of scalarized optimality, it is assumed that the decision maker is able to *weigh* the importance of each objective; the objective function vector is then *scalarized* according to some weight vector $\vec{\lambda} = (\lambda_1, \ldots, \lambda_Q)$. We will denote the scalarized objective function value by $f_{\lambda}(s)$.

We then say that a solution $s \in \mathcal{S}$ is a scalarized global optimum solution if and only if there is no $s' \in \mathcal{S}$ such that $f_{\lambda}(s) < f_{\lambda}(s')$ with respect to a given $\vec{\lambda}$. The weight vector $\vec{\lambda}$ is usually normalized such that $\sum_{q=1}^{Q} \lambda_q = 1$. Thus, $\vec{\lambda}$ is an element from the set of normalized weight vectors Λ given by

$$\Lambda = \{\vec{\lambda} \in R^Q : \lambda_q > 0, \sum_{q=1}^{Q} \lambda_q = 1, q = 1, \ldots, Q\} \tag{29.1}$$

In the case of different ranges of values between objectives, a normalization by range equalization factors must be considered [14]. The scalarization of the objective function vector is usually based on the family of weighted L_p-metrics as

$$f_{\lambda}(s) = \left[\sum_{i=1}^{Q} \left(\lambda_i | f_i(s) - y_i | \right)^p \right]^{1/p} \tag{29.2}$$

where $s \in \mathcal{S}$, $p > 0$, and $\vec{y} = y_1, \ldots, y_Q$ is the *ideal* vector, where we have $y_i = \min f_i$, $i = 1, \ldots, Q$. Settings of $p = 1$ or $p = \infty$ are most often used. When $p = 1$, we have the well-known *weighted sum* formulation given by

$$f_{\lambda}(s) = \sum_{i=1}^{Q} \lambda_i f_i(s) \tag{29.3}$$

It is well known that a scalarized global optimum solution for Eq. (29.3) with $p \neq \infty$ is also a Pareto global optimum solution, either if it is a unique solution or if the components of the weight vectors are all positive [14]. Obviously, the great advantage of using a weighted sum formulation is that the same SLS algorithm for solving the single-objective problem can be used for tackling the multiobjective version. When using such formulations, the available algorithms typically change the weight vectors to generate solutions that are of high quality for different weights. A disadvantage is that, when finding optimal solutions with respect to the weighted sum formulation, only *supported solutions*, that is, solutions on the convex hull of the efficient set, are obtained.

Finally, let us mention that (with a few exceptions) common to all SLS algorithms for MCOPs is that they return a set of nondominated solutions. Therefore, most of these algorithms include an additional data structure that maintains a set of solutions during the search process that we call *archive* and that is returned when the algorithm is terminated at an arbitrarily chosen time. In our review, we consider that the best nondominated solutions found during the algorithm's run are maintained in the archive, if not expressed otherwise. During the search process, the algorithm needs to *update* the archive and, if nothing else is said, we assume that this update consists of (i) adding new nondominated solutions and (ii) removing dominated ones.

Many of the available SLS algorithms make use of two further techniques. The first is *archive bounding*; it is used because the archive may grow very strongly and the operations for manipulating the archive become increasingly time consuming. The second are techniques for maintaining the solutions in the archive spread in the objective space, since it is assumed that clusters of solutions are not informative for the decision maker.

29.3 Component-Wise Acceptance Criterion

We first give an overview of SLS algorithms that make direct or indirect use of the component-wise ordering when deciding about the acceptance of new candidate solutions. By *direct* we mean that this decision is exclusively based on the component-wise ordering introduced in Section 29.2; by *indirect* we understand that from this component-wise ordering some *ranking* of candidate solutions is derived that is then finally used for deciding on which solutions to accept or choose for further manipulation. The algorithms that fall into this latter category are mainly evolutionary algorithms.

29.3.1 Direct Use of the Component-Wise Ordering

When designing an SLS algorithm for single-objective problems, one typically starts by implementing some form of an iterative improvement algorithm. In fact, it is relatively straightforward to apply iterative improvement algorithms also to multiobjective problems by modifying the acceptance criterion, making use of the component-wise ordering of solutions, and the use of an archive of nondominated solutions found so far. With such modifications, iterative improvement algorithms under the CWAC search model can iteratively improve the current set of candidate solutions in the archive by adding nondominated neighboring solutions to it [15]. Such an algorithm can be seeded either by one single solution that may be generated randomly, or by a set of candidate solutions generated by, for example, an exact algorithm. Despite their widespread use for single-objective problems, iterative improvement algorithms for MCOPs were proposed only recently; nevertheless, almost all SLS algorithms that make direct usage of the component-wise ordering are such iterative improvement algorithms or extensions thereof.

Iterative Improvement Algorithms

Among the first such approaches is Pareto local search (PLS) by Paquete et al. [16–18]. Pareto local search applies iteratively the two following steps. First, it selects randomly one candidate solution s from the archive that has not been visited and examines all neighbors of s. Second, it adds all neighbors of s that are nonweakly dominated with respect to the archive. It stops when the neighborhood of all candidate solutions has been examined. Independent of PLS, Angel et al. [19] proposed a similar approach called bicriteria local search (BLS). The main difference between PLS and BLS is that the latter examines the neighborhood of all candidate solutions in the archive, while PLS chooses only one and updates the archive immediately after examining a candidate solution's neighborhood. Angel et al. also proposed an extension of BLS by using an archive bounding technique that only accepts neighboring solutions whose objective function value vectors do not lie in a same partition of the objective space; in addition, also a restart version of BLS was presented. A similar idea was also proposed by Laumanns et al. [20] for a simple evolutionary multiobjective optimizer (SEMO) that examines only one randomly chosen solution in the neighborhood. A variant of SEMO [20], called fair evolutionary multiobjective optimizer (FEMO), selects the candidate solution of the archive whose neighborhood was examined least often.

An iterative improvement algorithm with a more complex acceptance criterion, called Pareto archived evolution strategy (PAES), was proposed by Knowles and Corne [21], which induces explicitly some dispersion among the solutions of a bounded-size archive. The acceptance criterion of PAES works as follows: A neighboring candidate solution s' is chosen randomly from the neighborhood of a current candidate solution s of the archive; if s' is dominated by any of the candidate solutions in the archive it is discarded, while if s' dominates candidate solutions in the archive, s' is added and the dominated candidate solutions are removed. If s' is nondominated with respect to s and to the archived candidate solutions, one of the two following possibilities is applied: (i) if the archive is not full, s' is added to the archive; (ii) if the archive is already full, s' is only added if there exists another solution s^* lying in a partition of the objective space that contains more solutions. In that case, s^* is removed from the archive. A more complex version of PAES, called M-PAES, is proposed in Ref. [22].

Extensions of Tabu Search

An obvious further extension of the iterative improvement algorithms described above is to incorporate features of other general-purpose SLS methods. Few approaches in that direction have been proposed so far and the ones we are aware of are based on multiobjective tabu search. The central idea is to examine the neighborhood of a set of solutions, extract nondominated solutions and accept among those only some nontabu ones for inclusion into an archive. Examples of such an approach have been presented by Baykasoglu et al. [23,24] and by Armetano and Arroyo [25]. In the latter approach, a bounded-size archive is used and special bounding and dispersion techniques are used that are based on the location of centroids of clusters of solutions in the objective space.

29.3.2 Indirect Use of the Component-Wise Ordering

Many current population-based SLS algorithms rely on a mapping of the objective function value vector of each candidate solution in the archive into a single value, a *rank*, where the lower a candidate solution's rank is, the higher are its chances of being chosen. This indirect use of the component-wise ordering is mainly made by a number of *multiobjective evolutionary algorithms* (MEAs). Here, we describe the most relevant of these MEAs with particular emphasis on the ranking procedure. All these approaches consider an archive of bounded size. An illustration of the ranking procedure used in the algorithms presented next is given in Figure 29.1; the brighter the color of an objective function value vector the better is the solution considered by the ranking procedure. Note that, as said before, we do not discuss details of these approaches such as crossover or mutation operators, since these are problem specific.

- Fonseca and Fleming [26] proposed the multiple objective genetic algorithm (MOGA), which scores each solution with the number of solutions that weakly dominate it in the archive. Then, solutions are ranked according to those values, where ties result in ranks being averaged. A sampling algorithm chooses the next set of solutions to remain in the archive, even if some of them are dominated. The top-left plot in Figure 29.1 shows this ranking procedure, before averaging tied ranks.
- Srinivas and Deb [27] proposed the nondominated sorting genetic algorithm (NSGA), which extended a ranking procedure initially proposed (but not tested) by Goldberg [28]: the lowest rank is assigned to the set of candidate solutions in the archive that are nondominated. These solutions are then removed and the nondominated solutions among the remaining candidate solutions are assigned the next rank level. This procedure is iterated until no candidate solution remains to be assigned a rank. Next, a sampling procedure is used for choosing the next set of solutions according

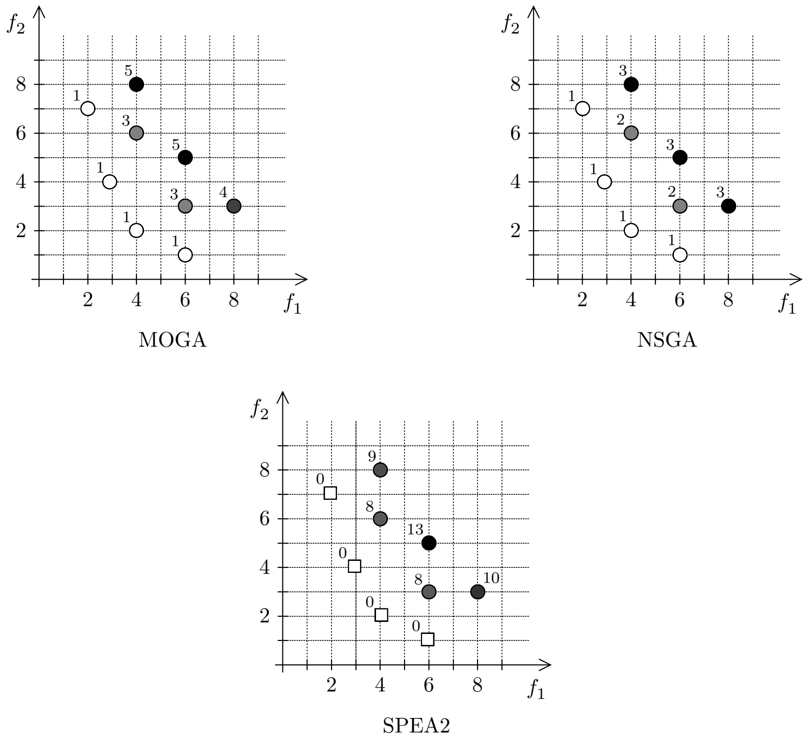


FIGURE 29.1 Graphical illustration of several ranking procedures in MEAs where the numbers at the points indicate the respective rank values (See text for more details).

to the ranking. Similarly to MOGA, also dominated solutions could be chosen. The top-right plot in Figure 29.1 illustrates this ranking procedure. Some further improvements are found in NSGA-II [29]; these include a faster computation of the ranks and a strategy for maintaining a dispersed set of solutions called *crowding*.

- Zitzler et al. [30] proposed strength Pareto evolutionary algorithm 2 (SPEA2) that maintains two sets of solutions, one being the archive of the best nondominated solutions found so far and the second being the set of current candidate solutions that play the usual role of the population in evolutionary algorithms. Some of the solutions from the archive might be removed by a clustering algorithm if their number exceeds the maximum allowable size. The ranking procedure of SPEA2 works as follows:
  1. To each solution in the archive and in the set of current solutions is assigned the number of current solutions that are dominated by it.
  2. For each solution in both sets, its rank is given by the sum of the values computed for the solutions from both sets that weakly dominate it.

  Solutions, whose ranks are less than one, are added to the archive. The size of the archive is maintained fixed at some value j in two ways: if the number of solutions becomes larger than j, solutions which are clustered in the objective space are removed (except those that are the best to each objective); if the archive is not full, the best current solutions are added to it until the total number of solutions reaches j. The bottom plot in Figure 29.1 illustrates this ranking procedure, where circles correspond to the set of current solutions and squares correspond to the solutions in the archive. The ranking procedure used by SPEA2 corrects the previous version called SPEA [31], in which a solution that is dominated by another one could be assigned a lower rank.

Since these approaches typically use an archive of fixed size, it is desirable that the existing solutions are dispersed in the objective space. Therefore, this aspect also affects how solutions are going to be chosen. Both MOGA and NSGA use a *fitness sharing* strategy, which decreases a solution's rank depending on how large is the number of solution in the archive within a certain radius in the objective space. NSGA-II applies a *crowding* strategy, which modifies a solution's rank based on an average distance from the nearest nondominated solutions in the objective space. A similar approach is also used in SPEA2, where to each solution rank the inverse of the distance from the kth nearest neighbor solution in the objective space is added (k is a parameter).

Recently, some MEAs have been proposed that add a further exploration step by making direct use of the component-wise ordering. For instance, Talbi [32] proposed an algorithm that starts with an MEA, whose final set of solutions is used afterward as starting solutions for an iterative improvement similar to PLS; similar approaches are found also in Brizuela et al. [33] and Basseur et al. [34]. In Jozefowiez et al. [35], the further step after the termination of an MEA consists of a tabu search algorithm, with similar principles to Ref. [23], that is run several times for different regions of the objective space. Finally, Morita et al. [36] proposed a more complex combination by maintaining two sets of solutions such as SPEA2. At each iteration, some solutions are chosen from each set and are recombined and mutated; the resulting solutions are then added to the current set of solutions and to the archive. Then, the CWAC step consists of the examination of the solutions neighboring to one chosen from the archive. Finally, the archive is updated and some solutions from the current set of solutions are removed to maintain a given cardinality at the end of each iteration.

29.4 Scalarized Acceptance Criterion

The main principle underlying the SAC search model is to use the value returned by the scalarization of the objective function vector with respect to some weight vector to distinguish between *better* and *worse* solutions. Obviously, using only one weight vector is not enough for obtaining a reasonable approximation to the efficient set. Hence, most approaches consider to change the components of the weight vector while running the algorithm to attain different regions of the objective space.

In the following, we divide the approaches in *nonproprietary* and *proprietary* ones. In nonproprietary approaches, an SLS algorithm is embedded into a general framework that mainly says how an underlying SLS algorithm is applied to tackle an MCOP. In the proprietary approaches, some specific, general-purpose SLS method is enhanced by additional features or specific search strategies that make it adapted for tackling MCOPs. Several examples of such proprietary approaches are therefore discussed in dependence of the underlying general-purpose SLS method.

29.4.1 Nonproprietary Approaches

In the simplest case, a single-objective SLS algorithm could be run several, say k, times using k different weight vectors and one could for each scalarization return the best solution found by the SLS algorithm. Then, the set of objective value vectors of the k returned final solutions forms an approximation to the efficient set. To output a set of nondominated solutions, the dominated solutions from the final set are removed.

Surprisingly, such a very basic approach is very rarely used, not even for a comparison to more complex algorithms. Exceptions are found in Borges and Hansen [37] and Knowles and Corne [38], who used the set of solutions returned by such an approach to get insight into certain instance features. In the following, we describe some approaches that further extend these ideas:

- Borges [39] proposed a general framework, called changing horizon efficient set search (CHESS), whose acceptance criterion is based on a function of the distance between a new solution and an archive; in particular, the neighboring solution that is accepted to the archive is the one that maximizes the minimum difference between each component of the objective function value vector to any solution in the archive. However, this distance does not take into account any weight vector. This general rule for the acceptance criterion can be applied to SLS methods such as simulated annealing or tabu search.
- Paquete and Stützle [40] proposed two-phase local search (TPLS), which works as follows: in a first phase, a high-quality solution for one objective is obtained by some high-performance SLS algorithm. Then, in the second phase, a sequence of scalarizations is solved; the initial solution of each scalarized problem is the one returned by the previous scalarization; the starting solution for the first scalarization is the one returned from the first phase. The weight vectors that define the scalarizations in the second phase are modified according to some strategy. This strategy could consist of a random sequence of weight vectors, or of a sequence such that a small change is incurred between components of successive weight vectors. This latter strategy is the most applied one so far. For a detailed description of this algorithm see Refs. [18,40,41].

29.4.2 Proprietary Approaches

Proprietary Simulated Annealing

Differently from the CWAC model, many algorithms that follow the SAC model use simulated annealing principles. Here, we describe some of the most relevant ones:

- Serafini [42] proposed several ways of modifying the usual probabilistic acceptance criterion of simulated annealing for tackling multiobjective problems. Given the current solution s and a neighboring solution s', he gives guidelines that should be applied to the computation of the probability p of accepting s': if $\vec{f}(s') \prec \vec{f}(s)$, then $p = 1$; if $\vec{f}(s) \prec \vec{f}(s')$, then $p < 1$; otherwise, p depends on the value returned from a function of a parameter called "temperature" and the weighted distance between $\vec{f}(s')$ and $\vec{f}(s)$ (or between $\vec{f}(s')$ and the ideal vector). To attain more solutions, Serafini proposed to use small random variations on the components of the weight vector during the run.
- Ulungu [43] proposed Multi-Objective Simulated Annealing (MOSA), where a set of weight vectors is defined a priori and for each scalarization one run of a simulated annealing algorithm is done. The

probabilistic acceptance criterion in MOSA follows similar principles to those proposed by Serafini [42]. Each time a neighboring solution is accepted, an archive A_λ of nondominated solutions for the current weight vector λ is updated; the final set of solutions returned by the algorithm is obtained after removing the dominated solutions from the union of the resulting sets A_λ.

Proprietary Tabu Search

Hansen [44] proposed Multi-Objective Tabu Search (MOTS) that uses an archive, which is improved during the search process. In MOTS, only neighboring solutions that are nontabu and the best with respect to a given scalarization can be added to the archive. To obtain a final set of solutions that is dispersed in the objective space, the current weight vector is updated such that neighboring solutions that are isolated in the objective space are preferably chosen. This update is done as follows: given a solution s from the archive, the ith weight vector component increases by a fixed amount for each solution in the archive that is worse in the ith objective; then, a *nontabu* neighbor of s that has the best scalarized objective function value with respect to the new weight vector is chosen and added to the archive if it is nondominated with respect to all solutions in the archive and, in that case, the tabu list is updated. Further features of MOTS are described in Ref. [44].

Proprietary Memetic Algorithms

Several memetic algorithms for MCOPs have been proposed in analogy to principles of the single-objective case. Usually, these algorithms consist of a sequence of runs of an SLS algorithm using a scalarized objective function, each one seeded by solutions that are generated by recombination and mutation procedures applied to elements of the current set of solutions. We describe the following main approaches:

- Ishibuchi and Murata [45] proposed Multi-Objective Genetic Local Search (MOGLS), which is a straightforward extension of MEAs to memetic algorithms. At each iteration, two solutions are selected from the archive by a sampling procedure that takes into account the ranking of all solutions based on a scalarization of the objective function vector with respect to a randomly generated weight vector. These two solutions are then recombined, generating a new one, which is then further improved by a local search algorithm that uses the current weight vector. In the first proposal only one randomly chosen solution from the neighborhood is examined at each iteration of the local search, but more recently also analyses on the influence of the strength of the local search on the overall performance have been done [46].
- Jaszkiewicz [47] proposed another memetic algorithm also called MOGLS. This algorithm maintains two sets of solutions as SPEA2, where one set holds the current solutions while a second one corresponds to the archive of the best solutions found. At each iteration, two solutions are taken from a subset of the best current solutions with respect to a certain randomly chosen weight vector and then recombined. Next, a local search algorithm starts from this new solution using the scalarized objective function defined by the generated weight vector. The solution returned by the local search algorithm is then added to the two sets of solutions according to some acceptance rules.

29.5 Combination of Search Models

Several recently proposed algorithms combine the SAC and the CWAC search model, giving place to hybrid algorithms. Two main trends can be identified. Either the overall SLS algorithm applies two clearly distinct phases based on the two search models in sequence (sequential combination), or the SLS algorithm combines components of the two search models and iteratively changes between those in the overall search process (iterative combination).

29.5.1 Sequential Combination

All the sequential combinations we are aware of first apply algorithms based on the SAC search model. The reason for this choice may be due to the fact that optimal solutions with respect to scalarizations of the objective function vector can identify only supported solutions, possibly leaving large gaps in the final set of solutions returned; in addition, the number of solutions returned by some methods following the SAC model is limited to the number of scalarizations. Algorithms that follow the CWAC search model can easily be applied after the SAC step with the aim of filling these gaps and increasing the number of returned solutions. Note that in the SAC step of such a combination, not necessarily SLS algorithms are required. In fact, if the scalarized problem is polynomially solvable, it is more useful to use an exact algorithm; this is also the case in the first two combinations described below.

- Hamacher and Ruhe [48] and Andersen et al. [49] proposed an algorithm that combines the two search models for tackling the multiobjective minimum spanning tree problem. In their approaches, the SAC step consists of obtaining several supported solutions for different scalarizations (note that a minimum spanning tree can be computed in polynomial time) and in the CWAC step, candidate solutions in the neighborhood of the supported ones are added to the archive if they are nondominated by any other candidate solution.
- Gandibleux et al. [50] proposed an algorithm to search for nondominated solutions with respect to a set of supported solutions that was previously obtained in the SAC step. (As in Refs. [48,49], an efficient algorithm is known to the single-objective version of the MCOP tackled in that paper, the multiobjective assignment problem.) The CWAC step follows principles from evolutionary algorithms: several pairs of solutions are chosen randomly from the archive and are recombined to generate a new set of solutions; the recombination operator used here takes into account some information about the components of the supported solutions. The so obtained solutions are then changed by a mutation procedure and, if the mutated solutions are not dominated by some previously calculated bounds, their neighborhood is explored and the nondominated neighbors are added to the archive. This step is repeated for a given number of iterations.
- Paquete and Stützle [40] proposed a Pareto double two-phase local search (PDTPLS) as a further extension of the TPLS approach (see Section 29.4.1). For each scalarization, a solution s is returned and a CWAC step examines all neighboring solutions of s; all nondominated solutions found in this way are returned once all scalarizations have been examined.

29.5.2 Iterative Combination

Iterative combinations of the SAC and CWAC search models could be conceived in various ways and, hence, it is probably not surprising that the approaches in this category are more varied than when only sequential combinations are considered. In the following we restrict ourselves to give some examples of such combinations that range from local search algorithms like tabu search and simulated annealing to population-based algorithms like memetic algorithms or ACO algorithms. These examples illustrate the range of possibilities that are opened by such iterative combinations.

- Gandibleux et al. [51] use tabu search principles in an algorithm called MOTS. This algorithm iteratively changes between the neighborhood exploration based on an SAC search model and CWAC search model. In the SAC step, a nontabu solution that minimizes the distance from a *local* utopian point[1] or a tabu but aspired solution replaces the current one. Once a new solution in the neighborhood of the current one, s, is accepted, a CWAC step adds nondominated neighbors of s to the archive. To favor more isolated regions of the objective space, the weight vector is updated periodically; to maintain the diversity of the search process, weight vectors are declared tabu for a given number of iterations.
- Abdelaziz and Krichen [52] developed an algorithm that was also called MOTS. Given some solution s, the SAC step works as follows: A constructive algorithm is run several times for several scalarizations to generate a set T of solutions; for each run of the constructive algorithm, the weight

[1] An utopian point dominates the ideal point. In MOTS, a local utopian point dominates all neighbors of the current solution.

vector takes into account the two worst components of the objective function value vector of s multiplied by some random values. The neighboring solutions to s are added to T, from which then the dominated solutions are removed and the resulting set T' is used to update the archive. Then, a CWAC step that follows similar principles as in Ref. [23] (see also Section 29.3.1) is applied for a given number of iterations. The iterative combination of the SAC and CWAC steps is repeated until no solution can be added to the archive during some iterations. A further extension, using an MEA, has been proposed in Ref. [53].

- Czyzak and Jaszkiewicz [54] proposed an algorithm which further extends the simulated annealing principles for tackling MCOPs. First, a set of solutions T is generated randomly and each solution from this set is assigned a weight and added to the archive. For each candidate solution s from T, a neighboring solution s' is added to the archive if it is not dominated by s (CWAC step). The SAC step is then applied for determining s' as the new current solution according to probabilistic rules similar to the ones proposed by Serafini; the weight vector is changed automatically during the search process to favor the acceptance of neighboring solutions that are more *isolated* in the objective space. This iterative process is repeated for all solutions in T.
- López-Ibáñez et al. [55] proposed a combination of SPEA2 with SLS algorithm, which combines a CWAC step, as executed by SPEA2, with a SAC step used for defining the search direction for the SLS algorithm. In that proposal, the SLS algorithm is applied to every new solution obtained from a recombination of solutions in the archive using randomly generated weights.

Recently, there has been some interest in applying ACO algorithms [56] to MCOPs. Ant colony optimization algorithms are based on the repeated, construction of solutions that is stochastically biased by artificial *pheromone* trails (pheromone trails are essentially some numerical information attached to solution components that are used in the solution construction) and heuristic information on the problem under being solved. The pheromone trails are updated during the algorithm's run in dependence of the search experience. Interestingly, all applications of ACO algorithms to MCOPs that are solved under the notion of Pareto optimality combine the two search models: The solution construction is typically based on an SAC step, where weights are defined to join various types of pheromone information with respect to the various objectives, while the artificial ants (representing the generated solutions) that update the pheromone information are typically chosen based on a CWAC step. Some approaches are described as follows:

- Iredi et al. [57] proposed several extensions of ACO algorithms where the pheromone and heuristic information are associated to different weight vectors, defined according to subintervals within the range [0, 1]. Thus, different weight vectors direct the constructive steps toward different regions of the objective space. The artificial pheromones are then updated with the nondominated solutions found. Extensions of these algorithms were also proposed by López-Ibáñez et al. [55,58], where an SLS algorithm based on a scalarization of the objective function vector is applied after each new constructed solution.
- Doerner et al. [59] describe an ACO algorithm that uses one pheromone matrix for each objective but only one same type of heuristic information. Each solution is constructed based on the heuristic information and on the weighted aggregation of the pheromone matrices according to a randomly chosen weight vector. Some of the best nondominated solutions found are then used for updating the pheromones.

29.6 Conclusions

We have reviewed the research on SLS algorithms for tackling MCOPs with respect to the notion of Pareto optimality. The existing approaches were classified according to whether they use the SAC and CWAC search models or some combinations thereof. For each of the resulting main classes of algorithms we have shortly described some main representatives without the intention of providing a fully comprehensive enumeration of all the existing proposals, which would be almost impossible, given their large number and the limited amount of space. While we described here only the main features of available approaches, several other interesting details can be found in the original papers.

Among the possible approaches for tackling MCOPs, a recent and interesting one is to use an acceptance criterion based on solution quality indicators. Such an algorithm has been proposed by Knowles et al. [60]; it applies at each step the hypervolume indicator [31] to decide which solutions are added to the archive. A similar approach has also been followed in Ref. [61], where binary performance indicators are used, that is, quality indicators that return a pair of values for each nondominated set. We remark that the performance of these algorithms strongly depends on the performance of the underlying algorithm for computing the quality indicators and we expect further development on ways of computing these more efficiently.

There are a few important areas related to SLS algorithms for MCOPs that were not covered here. One of the most important is the empirical assessment of the performance of these algorithms. This is far from being a trivial issue since, as shown by Zitzler et al. [62], frequently used quality indicators for comparing nondominated sets obtained by SLS algorithms have severe problems. A recognized exception is the use of attainment functions. Initially proposed in Ref. [63], the attainment function characterizes the performance of the SLS algorithms by describing the distribution of the outcomes. This function has shown to be a first-order moment measure of these outcomes and it can be seen as generalization of the multivariate empirical distribution function [64]. This allows the use of statistical inference and experimental design techniques to *infer* conclusions on the performance of SLS algorithms [65,66]. Some further extension of attainment functions for second-order moments can be found in Ref. [67].

Finally, we remark that little is known on the dependence between the performance of SLS algorithms and certain features of the MCOPs. Exceptions can be found in Mote et al. [68] and Müller-Hannemann and Weihe [69] who identified several features of the Multiobjective Shortest Path problem and variations that translate in a tractable number of Pareto optimal solutions, which contrasts with the known worst case for the same problem; see also Ref. [70]. In Refs. [17,37], it is conjectured that most efficient solutions and approximations thereof are strongly clustered in the solution space for the multiobjective traveling salesman problem with respect to a small-sized neighborhood. This means that local search algorithms using this neighborhood are very suitable for this problem, which is confirmed in Ref. [40]. In addition, the correlation between objectives seems to have a strong effect on the choice for the search model; in Ref. [18], experimental results indicate that simple SLS algorithms based on the SAC and CWAC search model, respectively, can behave very differently for the multiobjective quadratic assignment problem as the correlation between objectives is changed. Therefore, an interesting future line of research is to investigate, both experimentally and analytically, which and how instance features affect the performance of SLS algorithms.

While the research field of applying SLS algorithms to MCOPs poses many open questions and significant research issues, the set of available SLS algorithms shows that nowadays it is becoming increasingly feasible to tackle MCOPs with respect to the notion of Pareto optimality and that, therefore, these approaches are very likely to receive more attention for the solution of difficult real-world multiobjective problems.

Acknowledgments

This work was supported in part by the “Metaheuristics Network,” a Research Training Network funded by the Improving Human Potential programme of the CEC. Thomas Stützle acknowledges the support of the Belgian FNRS, of which he is a research associate.

The information provided is the sole responsibility of the authors and does not reflect the Community’s opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] Coello, C. C., Aguirre, A. H., and Zitzler, E., Eds., *Evolutionary Multi-Criterion Optimization*, Lecture Notes in Computer Science, Vol. 3410, Springer, Berlin, 2005.

[2] Fonseca, C. M., Fleming, P., Zitzler, E., Deb, K., and Thiele, L., Eds., *Evolutionary Multi-Criterion Optimization*, Lecture Notes in Computer Science, Vol. 2632, Springer, Berlin, 2003.

[3] Gandibleux, X., Sevaux, M., Sörensen, K., and T'kindt, V., Eds., *Metaheuristics for Multiobjective Optimisation*, Lecture Notes in Economics and Mathematical Systems, Vol. 535, Springer, Berlin, 2004.

[4] Zitzler, E., Deb, K., Thiele, L., Coello, C. C., and Corne, D., Eds., *Evolutionary Multi-Criterion Optimization*, Lecture Notes in Computer Science, Vol. 1993, Springer, Berlin, 2001.

[5] Coello, C. C., Special issue on "evolutionary multiobjective optimization," *IEEE Trans. Evol. Comput.*, 7(2), 2003.

[6] Gandibleux, X., Jaszkiewicz, A., Fréville, A., and Slowinski, R., Special issue on "multiple objective metaheuristics," *J. Heuristics*, 6(3), 2000.

[7] Jaszkiewicz, A., Special issue on "evolutionary and local search heuristics in multiple objective optimization," *Found. Comput. Decision Sci. J.*, 26(1), 2001.

[8] Coello, C. C., A comprehensive survey of evolutionary-based multiobjective optimization techniques, *Knowl. Inf. Syst.*, 1, 269, 1999.

[9] Coello, C. C., An updated survey of GA-based multiobjective optimization techniques, *ACM Comput. Surv.*, 32, 109, 2000.

[10] Ehrgott, M. and Gandibleux, X., Approximative solution methods for combinatorial multicriteria optimization, *TOP*, 12(1), 1, 2004.

[11] Fonseca, C. M. and Fleming, P., An overview of evolutionary algorithms in multiobjective algorithms, *Evol. Comput.*, 3, 1, 1995.

[12] Knowles, J. and Corne, D., Memetic algorithms for multiobjective optimization: issues, methods and prospects, in *Recent Advances in Memetic Algorithms*, Vol. 166, Krasnogor, N., Smith, J., and Hart, W., Eds., Studies in Fuzziness and Soft Computing, Springer, Heidelberg, 2004, p. 313.

[13] Jones, D., Mirrazavi, S., and Tamiz, M., Multi-objective meta-heuristics: an overview of the current state-of-the-art, *Eur. J. Oper. Res.*, 137(1), 1–9, 2002.

[14] Steuer, R. E., *Multiple Criteria Optimization: Theory, Computation and Application*, Wiley, New York, 1986.

[15] Paquete, L., Schiavinotto, T., and Stützle, T., On Local Optima in Multiobjective Combinatorial Optimization Problems, TR AIDA-04-11, FG Intellektik, TU Darmstadt, 2004.

[16] Paquete, L., Chiarandini, M., and Stützle, T., A Study of Local Optima in the Multiobjective Traveling Salesman Problem, TR AIDA-02-07, Fachgebiet Intellektik, Fachbereich Informatik, Technische Universität Darmstadt, 2002.

[17] Paquete, L., Chiarandini, M., and Stützle, T., Pareto local optimum sets in the biobjective traveling salesman problem: an experimental study, in *Metaheuristics for Multiobjective Optimisation*, Lecture Notes in Economics and Mathematical Systems, Vol. 535, Gandibleux, X., Sevaux, M., Sörensen, K., and T'kindt, V., Eds., Springer, Berlin, 2004, p. 177.

[18] Paquete, L. and Stützle, T., A study of stochastic local search algorithms for the biobjective QAP with correlated flow matrices, *Eur. J. Oper. Res.*, 169(3), 943, 2006.

[19] Angel, E., Bampis, E., and Gourvés, L., A dynasearch neighborhood for the bicriteria traveling salesman problem, in *Metaheuristics for Multiobjective Optimisation*, Lecture Notes in Economics and Mathematic Systems, Vol. 535, Gandibleux, X., Sevaux, M., Sörensen, K., and T'kindt, V., Eds., Springer, Berlin, 2004, p. 153.

[20] Laumanns, M., Thiele, L., Zitzler, E., Welzl, E., and Deb, K., Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem, *Proc. of PPSN-VII*, Lecture Notes in Computer Science, Vol. 2439, Springer, Berlin, 2002, p. 44.

[21] Knowles, J. and Corne, D., The Pareto archived evolution strategy: a new base line algorithm for multiobjective optimisation, *Proc. Congress on Evolutionary Computation,* IEEE Press, Piscataway, NJ, 1999, p. 98.

[22] Knowles, J. and Corne, D., M-PAES: a memetic algorithm for multiobjective optimization, *Proc. Congress on Evolutionary Computation,* Vol. 1, IEEE Press, Piscataway, NJ, 2000, p. 325.

[23] Baykasoglu, A., Owen, S., and Gindy, N., A taboo search based approach to find the Pareto optimal set in multiobjective optimization, *J. Eng. Opt.*, 31, 731, 1999.

[24] Baykasoglu, A., Özbakir, L., and Sönmez, A., Using multiple objective tabu search and grammars to model and solve multi-objective flexible job shop scheduling problems, *J. Intell. Manuf.*, 15, 777, 2004.

[25] Armetano, V. A. and Arroyo, J. E., An application of a multi-objective tabu search algorithm to a bicriteria flowshop problem, *J. Heuristics*, 10(5), 463, 2004.

[26] Fonseca, C. M. and Fleming, P., Genetic algorithms for multiobjective optimization: formulation, discussion and generalization, *Proc. Int. Conf. on Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, 1993, p. 416.

[27] Srinivas, N. and Deb, K., Multiobjective optimization using nondominated sorting in genetic algorithms, *Evol. Comput.*, 3(2), 221, 1994.

[28] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[29] Deb, K., Agrawal, S., Prataand, A., and Meyarivan, T., A fast and elitist multi-objective genetic algorithm: NSGA-II, *IEEE Trans. Evol. Comput.*, 6(2), 182, 2002.

[30] Zitzler, E., Laumanns, M., and Thiele, L., SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization, TR TIK-Report 103, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, May 2001.

[31] Zitzler, E. and Thiele, L., Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, *IEEE Trans. Evol. Comput.*, 4(3), 257, 1999.

[32] Talbi, E. G., A hybrid evolutionary approach for multicriteria optimization problems: application to the flow shop, in *Evolutionary Multi-Criterion Optimization*, Vol. 2632, Fonseca, C. M., Fleming, P., Zitzler, E., Deb, K., and Thiele, L., Eds., Lecture Notes in Computer Science, Springer, Berlin, 2003, p. 416.

[33] Brizuela, C. A., Sannomiya, N., and Zhao, Y., Multi-objective flow-shop: preliminary results, in *Evolutionary Multi-Criterion Optimization*, Vol. 1993, Zitzler, E., Deb, K., Thiele, L., Coello, C. C., and Corne, D., Eds., Lecture Notes in Computer Science, Springer, Berlin, 2001, p. 443.

[34] Basseur, M., Seynhaeve, F., and Talbi, E. G., Design of multi-objective evolutionary algorithms: application to the flow-shop, *Proc. Congress on Evolutionary Computation*, Vol. 2, IEEE Press, Piscataway, NJ, 2002, p. 151.

[35] Jozefowiez, N., Semet, F., and Talbi, E.-G., Parallel and hybrid models for multi-objective optimization: application to the vehicle routing problem, *Proc. Int. Conf. on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, Vol. 2439, Springer, Berlin, 2002, p. 271.

[36] Morita, H., Gandibleux, X., and Katoh, N., Experimental feedback on biobjective permutation scheduling problems solved with a population heuristic, *Found. Comput. Decision Sci.*, 26(1), 23, 2001.

[37] Borges, P. C. and Hansen, M. P., A study of global convexity for a multiple objective travelling salesman problem, in *Essays and Surveys in Metaheuristics*, Ribeiro, C. C. and Hansen, P., Eds., Kluwer Academic Publishers, Boston, MA, USA, 2000, p. 129.

[38] Knowles, J. and Corne, D., Towards landscape analyses to inform the design of a hybrid local search for the multiobjective quadratic assignment problem, *Soft Computing Systems Design, Management and Applications*, IOS Press, Amsterdam, 2002, p. 271.

[39] Borges, P., CHESS—changing horizon efficient set search: a simple principle for multiobjective optimization, *J. Heuristics*, 6(3), 405, 2000.

[40] Paquete L. and Stützle, T., A two-phase local search for the biobjective traveling salesman problem, in *Evolutionary Multi-Criterion Optimization*, Vol. 2632, Fonseca, C. M., Fleming, P., Zitzler, E., Deb, K., and Thiele, L., Eds., Lecture Notes in Computer Science, Springer, Berlin, 2003, p. 479.

[41] Paquete, L., Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: Methods and Analysis, Ph.D. thesis, FB Informatik, TU Darmstadt, Germany, 2005.

[42] Serafini, P., Simulated annealing for multiobjective optimization problems, in *Multiple Criteria Decision Making. Expand and Enrich the Domains of Thinking and Application*, Tzeng, G. H., Wang, H. F., Wen, V. P., and Yu, P. L., Eds., Lecture Notes in Economics and Mathematic Systems, Springer, New York, 1994, p. 283.

[43] Ulungu, E. L., Optimisation combinatoire multicritére: Détermination de l'ensemble des solutions efficaces et méthodes interactives, Ph.D. thesis, Université de Mons-Hainaut, Mons, Belgium, 1993.

[44] Hansen, M. P., Tabu search for multiobjective optimisation: MOTS, *Proc. Int. Conf. on Multiple Criteria Decision Making*, 1997.

[45] Ishibuchi, H. and Murata, T., A multi-objective genetic local search algorithm and its application to flow-shop scheduling, *IEEE Trans. Syst., Man., Cybern.—Part C*, 28(3), 392, 1998.

[46] Ishibuchi, H., Yoshida, T., and Murata, T., Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling, *IEEE Trans. Evol. Comput.*, 7(2), 204, 2003.

[47] Jaszkiewicz, A., Genetic Local Search for Multiple Objective Combinatorial Optimization, TR RA-014/98, Institute of Computer Science, Poznań University of Technology, Poznań, Poland, 1998.

[48] Hamacher, H. V. and Ruhe, G., On spanning tree problems with multiple objectives, *Ann. Oper. Res.*, 52, 209, 1994.

[49] Andersen, K., Jörnsten, K., and Lind, M., On bicriterion minimal spanning trees: an approximation, *Comput. Oper. Res.*, 23(12), 1171, 1996.

[50] Gandibleux, X., Morita, H., and Katoh, N., Use of a genetic heritage for solving the assignment problem, in *Evolutionary Multi-Criterion Optimization*, Vol. 2632, Fonseca, C. M., Fleming, P., Zitzler, E., Deb, K., and Thiele, L., Eds., Lecture Notes in Computer Science, Springer, Berlin, 2003, p. 43.

[51] Gandibleux, X., Mezdaoui, N., and Fréville, A., A tabu search procedure to solve multiobjective combinatorial optimization problems, in *Advances in Multiple Objective and Goal Programming*, Vol. 455, Caballero, R., Ruiz, F., and Steuer, R., Eds., Lecture Notes in Economics and Mathematic Systems, Springer, Berlin, 1997, p. 291.

[52] Abdelaziz, F. B. and Krichen, S., A Tabu Search Heuristic for Multiobjective Knapsack Problems, TR RRR 28-97, Rutgers Center for Operation Research, Piscataway, New Jersey, 1997.

[53] Abdelaziz, F. B., Chaouachi, J., and Krichen, S., A hybrid heuristic for multiobjective knapsack problems, in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Voss, S., Martello, S., Osman, I., and Roucairol, C., Eds., Kluwer Academic Publishers, Boston, MA, 1999, p. 205.

[54] Czyzak, P. and Jaszkiewicz, A., Pareto simulated annealing—a metaheuristic technique for multiple objective combinatorial optimization, *J. Multi-Criteria Decision Anal.*, 7, 34, 1998.

[55] López-Ibáñez, M., Paquete, L., and Stützle, T., Hybrid population-based algorithms for the biobjective quadratic assignment problem, *J. Math. Model. Algorithms*, 5(1), 111–137, 2006.

[56] Dorigo, M. and Stützle, T., *Ant Colony Optimization*, MIT Press, Cambridge, MA, 2004.

[57] Iredi, S., Merkle, D., and Middendorf, M., Bi-criterion optimization with multi colony ant algorithms, in *Evolutionary Multi-Criterion Optimization*, Vol. 1993, Zitzler, E., Deb, K., Thiele, L., Coello, C. C., and Corne, D., Eds., Lecture Notes in Computer Science, Springer, Berlin, 2001.

[58] López-Ibáñez, M., Paquete, L., and Stützle, T., On the design of ACO for the biobjective quadratic assignment problem, *Proc. of ANTS*, Lecture Notes in Computer Science, Vol. 3172, Springer, Berlin, 2004, p. 214.

[59] Doerner, K., Gutjahr, W. J., Strauss, C., and Stummer, C., Pareto ant colony optimization: a metaheuristic approach to portfolio selection, *Ann. Oper. Res.*, 131, 79, 2004.

[60] Knowles, J., Corne, D., and Fleischer, M., Bounded archiving using the lebesgue measure, *Proc. Congress on Evolutionary Computation*, IEEE Press, Piscataway, NJ, 2003, p. 2498.

[61] Zitzler, E. and Künzli, S., Indicator-based selection in multiobjective search, *Proc. Int. Conf. on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, Vol. 3242, Springer, Berlin, 2004, p. 832.

[62] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and Grunert da Fonseca, V., Performance assessment of multiobjective optimizers: an analysis and review, *IEEE Trans. Evol. Comput.*, 7(2), 117, 2003.

[63] Grunert da Fonseca, V., Fonseca, C. M., and Hall, A., Inferential performance assessment of stochastic optimizers and the attainment function, in *Evolutionary Multi-Criterion Optimization*, Vol. 1993, Zitzler, E., Deb, K., Thiele, L., Coello, C. C., and Corne, D., Eds., Lecture Notes in Computer Science, Springer, Berlin, 2001, p. 213.

[64] Grunert da Fonseca, V. and Fonseca, C. M., A link between the multivariate cumulative distribution function and the hitting function for random closed sets, *Stat. Prob. Lett.*, 57(2), 179, 2002.

[65] Shaw, K. J., Fonseca, C. M., Nortcliffe, A. L., Thompson, M., Love, J., and Fleming, P. J., Assessing the performance of multiobjective genetic algorithms for optimization of a batch process scheduling problem, *Proc. 1999 Congress on Evolutionary Computation,* Vol. 1, IEEE Press, Piscataway, NJ, 1999, p. 34.

[66] Paquete, L., López-Ibáñez, M., and Stützle, T., Towards an empirical analysis of SLS algorithms for multiobjective combinatorial optimization problems by experimental design, *Proc. Intl. Conf. Metaheuristics*, Vienna, Austria, 2005, pp. 739–746.

[67] Fonseca, C. M., Grunert da Fonseca, V., and Paquete, L., Exploring the performance of stochastic multiobjective optimisers with the second-order attainment function, in *Evolutionary Multi-Criterion Optimization*, Vol. 3410, Coello, C. C., Aguirre, A. H., and Zitzler, E., Eds., Lecture Notes in Computer Science, Springer, Berlin, 2005, p. 250.

[68] Mote, J., Murthy, I., and Olson, D. L., A parametric approach to solving bicriterion shortest path problem, *Eur. J. Oper. Res.*, 53, 81, 1991.

[69] Müller-Hannemann, M. and Weihe, K., Pareto shortest paths is often feasible in practice, *Int. Workshop on Algorithm Engineering*, Lecture Notes in Computer Science, Vol. 2141, Springer, Berlin, 2001, p. 185.

[70] Hansen, P., Bicriterion path problems, in *Multiple Criteria Decision Making Theory and Application*, Vol. 177, Fandel, G. and Gal, T., Eds., Lecture Notes in Economics and Mathematic Systems, Springer, Berlin, 1979, p. 109.

30

Sensitivity Analysis in Combinatorial Optimization

David Fernández-Baca
Iowa State University

Balaji Venkatachalam
University of California, Davis

30.1 Introduction

Anyone formulating an optimization problem faces uncertainty in selecting the model parameters. Transportation costs, probabilities of different events, and budget constraints, among other parameters, are all unlikely to be known with any degree of accuracy. There may also be doubts about the structure of the model, such as whether to add or remove constraints. Understanding the effect of changes in parameter values and model structure on the optimum solution is the subject matter of *sensitivity analysis*. This chapter gives an overview of the main algorithmic ideas in sensitivity analysis for combinatorial optimization.

Research in sensitivity analysis started shortly after the development of the simplex method [1,2]. Since then, the literature has grown rapidly: as of 1997, over 1000 journal articles had been published on the subject [3]. To keep things manageable, this chapter focuses on *parametric analysis*, which studies problems whose structure remains fixed, but where cost coefficients vary continuously as a function of one or more parameters. The reader interested in more information on topics not covered here has other good sources to turn to. Starting points into the literature include Geoffrion and Nauss' classical survey of the field in the late 1970s [4], the Ph.D. thesis of Wagelmans [5], the collection of articles edited by Gal and Greenberg [3], and Greenberg's bibliography [6].

Parametric analysis is a rich field, whose applications extend beyond the examination of alternative scenarios. In fact, parametric problems arise as auxiliary problems in other areas, such as Lagrangian relaxation and minimum-ratio optimization (see Section 30.2). Parametric analysis should not, however, be confused with the study of *parameterized complexity* [7]. The latter deals with finding efficient algorithms for problems when one of the input parameters is fixed. These parameters are typically discrete (e.g., the maximum size of a vertex cover), and the goal is to obtain the best solution, not to study its stability.

More in the spirit of parametric analysis, but also beyond the scope of this chapter, is the study of *stability of approximation*, which considers how the complexity of a problem depends on a parameter whose variation alters the space of allowable instances (Chapter 31 and Ref. [8]). An example is analyzing how the approximability of the traveling salesman problem varies as intercity distances are allowed to deviate from the triangle inequality as a function of a continuous parameter [9].

This chapter consists of five parts, in addition to this section. Section 30.2 gives basic definitions and notation. Section 30.3 introduces the key issues in sensitivity analysis. Section 30.4 is devoted to general-purpose techniques for sensitivity analysis. Section 30.5 discusses selected results on sensitivity analysis of polynomially solvable problems. Section 30.6 is devoted to sensitivity analysis of NP-hard problems.

30.2 Preliminaries

This chapter deals with combinatorial optimization problems of the form

$$z^* = \min\{\mathrm{cost}(x) : x \in X\} \tag{30.1}$$

where $X \subset \mathbb{R}^n$ is the set of *feasible solutions* and *cost*, referred to as the *cost function* or the *objective function*, maps each feasible solution x to a positive real number. Problem (30.1) is called a 0/1 *problem* when $X \subseteq \{0, 1\}^n$. The cost function is *linear* if $cost(x) = \sum_{i=1}^{n} c_i \cdot x_i$. The cost function is a *bottleneck* (or *min–max*) function when $cost(x) = \max_{1 \leq i \leq n} c_i \cdot x_i$. The c_i's are the *coefficients* of the objective function and $c = (c_1, \ldots, c_n)$ is the *coefficient vector.*

The minimum spanning tree and traveling salesman problems can be formulated as 0/1 problems with linear cost functions. In both cases, the coefficient vector consists of the edge costs and x_i is 1 or 0 depending on whether or not edge i is chosen. Bottleneck versions of both problems can also be defined. For example, in the bottleneck traveling salesman problem, the cost of a tour is determined by its costliest edge. Unless stated otherwise, our discussion will center on optimization problems with linear cost functions.

For any problem of the form (30.1), one can define a corresponding *parametric cost problem* where the cost of each feasible solution depends on a *parameter vector* λ:

$$Z(\lambda) = \min\{cost(x, \lambda) : x \in X\} \tag{30.2}$$

Eq. (30.2) defines an infinite family of optimization problems of the form (30.1), one for each fixed λ. Function $Z(\lambda)$ is called the *optimum cost function.* An *evaluator* for problem (30.2) is an algorithm that, for any given λ, computes $Z(\lambda)$ and an optimum solution at λ.

The optimum cost function Z induces a decomposition $\mathcal{M}(Z)$ of the parameter space into maximal connected regions, such that within each of region A, there exists a feasible solution x that is optimum for every $\lambda \in A$. The *combinatorial complexity* of $\mathcal{M}(Z)$ is the size of the description of $\mathcal{M}(Z)$ as a function of the input size.

Unless stated otherwise, the coefficients of the objective function of the parametric problems considered here are linear in λ; that is, $c_i(\lambda) = c_{i0} + \sum_{j=1}^{d} c_{ij}\lambda_j$ for $i = 1, \ldots, n$. In this case, $Z(\lambda)$ is a piecewise linear concave function of λ (assuming X corresponds to any reasonable combinatorial optimization problem), since it is the lower envelope of the set of cost functions of the feasible solutions. Hence, $\mathcal{M}(Z)$ subdivides the parameter space into convex polyhedral regions; its combinatorial complexity is its total number of facets of dimensions 0 through d.

An important subcase is the one where the parameter vector λ is precisely the coefficient vector c. This occurs, for example, when studying the sensitivity of a minimum spanning tree or of an optimum traveling salesman tour to variations in the edge costs. The special case where $d = 1$ is also of interest. Here the optimum cost function Z consists of a concave sequence of straight-line segments and the points at which these segments meet are called *breakpoints*; the combinatorial complexity of $\mathcal{M}(Z)$ is proportional to the number of breakpoints.

In addition to their applications to sensitivity analysis, parametric problems arise as auxiliary problems in other contexts. Two important cases are discussed next.

Lagrangian relaxation [10,11] is a well-known technique to handle certain kinds of hard optimization problems. Specifically, consider an NP-hard problem of the form

$$z^* = \min\{cx : Ax = b, x \in X\} \tag{30.3}$$

that is polynomially solvable in the absence of constraints $Ax = b$. For instance, the minimum spanning tree problem with degree constraints (known to be NP-hard [12]) can be expressed in the form (30.3): $X$is the set of spanning trees and the degree constraints are given by a system of linear equalities.

The *Lagrangian function* for problem (30.3) is

$$Z_L(\lambda) = \min\{cx + \lambda(Ax - b) : x \in X\} \tag{30.4}$$

Observe that Eq. (30.4) defines a linear parametric problem. For example, the Lagrangian function for the degree-constrained minimum spanning tree yields a minimum spanning tree problem where edge weights are functions of a parameter vector. It can be shown that for all λ, $z^* \geq Z_L(\lambda)$. The best lower bound obtainable from Eq. (30.4) is therefore $\max_\lambda Z_L(\lambda)$. In practice, this can be a quite accurate estimate of the optimum solution to problem (30.3) [13,14].

Minimum-ratio optimization, also known as *fractional programming*, deals with problems of the form

$$z^* = \min\{f(x)/g(x) : x \in X\} \tag{30.5}$$

where $g(x) > 0$ for all $x \in X$. One setting for this question arises in graph problems where each edge has a cost and a profit and the goal is to find a subgraph with a certain property that minimizes the ratio of total cost to total profit. Two examples are the minimum-ratio spanning tree problem [15] and the minimum ratio cycle problem [16].

Define

$$Z_R(\mu) = \min\{f(x) - \mu \cdot g(x) : x \in X\} \tag{30.6}$$

where μ is a scalar parameter. It is well known (see, e.g., Ref. [15]) that $z^* = \max\{\mu : Z_R(\mu) > 0\}$. Observe that $Z_R(\mu)$ is a concave piecewise linear decreasing function of μ. For the weighted graph problems mentioned above, Eq. (30.6) defines a parametric problem where edge costs are decreasing linear functions of μ.

Problems with parametric constraint set are also of interest. An example is the knapsack problem with variable knapsack size, which can be expressed as

$$Z_K(\mu) = \max\left\{\sum_{i=1}^{n} c_i x_i : \sum_{i=1}^{n} s_i x_i \leq b + \mu b', x \in \{0,1\}^n\right\} \tag{30.7}$$

where μ is a scalar parameter.

For reasons of space, problems with parametric constraint set are not treated in much depth here. It should be noted that these problems generally behave quite differently from those with parametric objective function. Indeed, the optimum cost function is typically not continuous, as the reader may verify for function Z_K above. An exception to this rule is linear programming with parametric right-hand side, which, because of duality, has the same behavior as linear programming with parametric objective function (see Section 30.5.1).

30.3 Issues in Sensitivity Analysis

Sensitivity analysis questions can be formulated in different ways. The first important class of problems considered here are collectively referred to as *posterior* analysis problems. Here, an optimum solution $x^{(0)}$ for some parameter vector $\lambda^{(0)} = (\lambda_1^{(0)}, \ldots, \lambda_d^{(0)}) \in \mathbb{R}^d$ is known and the question is to determine the *stability* of $x^{(0)}$ with respect to parameter variation around $\lambda^{(0)}$. Some common notions of stability are defined next.

Definition 30.1

The lower tolerance *and the* upper tolerance *of parameter i at $\lambda^{(0)}$ are the largest values l_i and u_i such that $x^{(0)}$ is optimum for every λ' such that $\lambda'_i \in [\lambda_i^{(0)} - l_i, \lambda_i^{(0)} + u_i]$ and $\lambda'_j = \lambda_j^{(0)}$ for $j \neq i$.*

A related problem is *ray shooting*: Given a ray ρ originating at $\lambda^{(0)}$, the question is to find the point $\lambda^* \in \rho$ farthest from $\lambda^{(0)}$ such that $x^{(0)}$ is optimum for every $\lambda \in \rho$ between $\lambda^{(0)}$ and λ^*. By suitable reparametrization, ray shooting can be expressed as a question of computing tolerances.

Definition 30.2

The stability region *of $x^{(0)}$ at $\lambda^{(0)}$ is the largest connected subset F of $\mathbb{R}^d$ containing $\lambda^{(0)}$ such that $x^{(0)}$ is an optimum solution for all $\lambda \in F$.*

Observe that each region of $\mathcal{M}(Z)$ is a stability region of some feasible solution x. Thus, stability regions are convex polyhedral subsets of the parameter space. In the important special case where the parameter vector is precisely the coefficient vector, the stability regions are polyhedral cones that meet at the origin. This is because all solutions have cost 0 at the origin of the parameter space. Note also that, in the one-parameter case, computing tolerances and stability regions are equivalent problems.

Definition 30.3

The stability radius *of $x^{(0)}$ at $\lambda^{(0)}$ is the radius of the largest ball B such that $x^{(0)}$ is optimum for all $\lambda \in B$.*

The notion of a ball in the definition above depends on the metric used. A common choice is the L_∞ norm, where the distance between parameter vectors λ and λ' is given by $\max_i |\lambda_i - \lambda'_i|$.

The *robustness function* [17] provides an alternative way to measure the effect of multiparameter variation.

Definition 30.4

The robustness function *at $\lambda^{(0)}$, is the function $R : \mathbb{R} \to \mathbb{R}$ given by*

$$R(b) = \max \left\{ Z(\lambda^{(0)} + \delta) : \sum_{i=1}^{d} w_i \cdot \delta_i \leq b, \quad \delta_i \geq 0 \text{ for } i = 1, \ldots, n \right\} \tag{30.8}$$

where $b \geq 0$ is the budget, *$\delta = (\delta_i, \ldots, \delta_d)$ is the* increment vector, *and $w_i \geq 0$ is the unit cost of increasing parameter λ_i, for each $i \in \{1, \ldots, n\}$.*

Intuitively, the robustness function yields the maximum effect that a total weighted increase of the parameters within a given budget can have on the cost of the optimum solution.

The next problems do not depend on advance knowledge of an optimum solution at some parameter value; questions of this kind are traditionally referred to as *prior analysis* problems. The following notion has some relationships with computing tolerances.

Definition 30.5

A most vital variable *in a* 0/1 *combinatorial optimization problem is a variable x_i such that forcing x_i to equal zero increases the value of the optimum solution as much as possible.*

In the context of network optimization problems, such as the minimum spanning tree problem, the shortest path problem, or the traveling salesman problem, identifying most vital variables often translates into finding most vital edges (see Sections 30.5, 30.6, and Chapter 62).

Parametric search encompasses a broad class of problems whose goal is to locate a point λ^* in the parameter space, where the optimum cost function satisfies some specified property. For example, ray shooting is a parametric search problem, although it is not an instance of prior analysis. Lagrangian relaxation and minimum-ratio optimization lead to two basic parametric search problems: *maximization*, that is, locating $\lambda^* = \arg\max_\lambda Z(\lambda)$, and *root finding*, that is, solving $Z(\lambda) = 0$ for λ.

Inverse optimization is another parametric search problem. The input here is a *reference solution* $x^{(0)}$, and the problem is to locate a parameter vector $\lambda^{(0)}$ such that $x^{(0)}$ is optimal, or as close to optimum as possible, at $\lambda^{(0)}$. Examples of inverse parametric minimum spanning tree, shortest path, matching, and other optimal subgraph problems are given in Ref. [18]; applications to biological sequence alignment are discussed in Refs. [19,20].

The *construction problem* is to build a complete representation of $\mathcal{M}(Z)$. This is the most general problem considered here, since almost any sensitivity analysis question can be answered given $\mathcal{M}(Z)$. Clearly, the time required to solve the construction problem depends heavily on the combinatorial complexity of Z.

30.3.1 Stability of Approximate Solutions and Heuristics

Variants of all the issues outlined so far arise when considering approximate solutions or the behavior of heuristic algorithms. These questions are of particular interest when studying NP-hard problems (see Section 30.6).

Analogs to the notions of tolerance, stability radius, and stability region for ϵ-approximate solutions are straightforward to define: simply replace "optimum" by "ϵ-approximate" in Definitions 30.1, 30.2, and 30.3. As observed in Ref. [21], the stability region for an ϵ-approximate solution x has properties similar to those of an ordinary stability region; e.g., it is closed, convex, and polyhedral. Unlike ordinary stability regions, however, the intersection between the stability regions of two different ϵ-approximate solutions, $\epsilon > 0$, can have dimension d. Note also that, for $\epsilon_1 < \epsilon_2$, the region where x is ϵ_1-approximate is contained in the region where x is ϵ_2-approximate [21].

So far, only the stability of *solutions* has been discussed. However, it is also natural to enquire about the sensitivity of an *algorithm* to changes in parameter values (see Section 30.6.3). Suppose A is an algorithm (typically a heuristic) for some optimization problem and let $x^{(0)}$ be the solution returned by A at parameter vector $\lambda^{(0)}$. Then, the *upper tolerance* of parameter i for A at $\lambda^{(0)}$ is the largest value u_i such that $x^{(0)}$ is the solution returned by A for all λ' such that $\lambda'_i \in [\lambda_i^{(0)}, \lambda_i^{(0)} + u_i]$ and $\lambda'_j = \lambda_j^{(0)}$ for all $j \neq i$. Lower tolerances, stability regions, and radii are defined similarly.

30.4 General Techniques

Several general-purpose methods for parametric analysis problems are known, including methods for parametric search, construction of the space decomposition, and determining the combinatorial complexity of low-dimensional problems. These techniques are surveyed next.

30.4.1 Parametric Search

Four methods of parametric search are reviewed here: bisection search, Newton's method, gradient descent, and Megiddo's method. The focus is on one-parameter problems, where the cost of a solution is a linear function of a single scalar parameter μ and the goal is to locate a value μ^* satisfying certain properties.

Newton's method. The classic root-finding method of Newton directly applies to the problem of finding a zero of $Z(\mu)$, when Z is a concave decreasing function of μ: Assume that a parameter value $\mu^{(1)}$ and the optimum solution $x^{(1)}$ at $\mu^{(1)}$ are given. Furthermore, assume that $\mu^{(1)} \leq \mu^*$. To locate μ^*, proceed as follows. Let $\mu^{(2)}$ be the value such that $cost(x^{(1)}, \mu^{(2)}) = 0$. Now compute $v = Z(\mu^{(2)})$ and the optimum solution $x^{(2)}$ at $\mu^{(2)}$. Note that $v \geq 0$. If $v = 0$, stop and return $\mu^* = \mu^{(2)}$. Otherwise, set $\mu^{(1)} = \mu^{(2)}$ and $x^{(1)} = x^{(2)}$ and repeat the process. This procedure generates an increasing sequence of μ values, each of which corresponds to a different feasible solution. Thus, the number of evaluations required by Newton's method is at most equal to the number of breakpoints of Z. This naive bound can be improved substantially in certain applications. In fact, Newton's method yields the fastest-known algorithms for several parametric search problems, including certain minimum-ratio network flow and minimum-cut problems [22,23]. Newton's method can also be applied to other parametric search problems, including finding the maximizer of Z or ray shooting [24].

Bisection search. Suppose that the parametric search problem under consideration has an *oracle*; that is, a procedure that determines whether a given parameter value μ is less than or equal to the parameter value μ^* being sought. Suppose, additionally, that one is given an interval $\mathcal{I}$ known to contain μ^*. Bisection search locates μ^* by repeatedly halving $\mathcal{I}$, taking the left or right half depending on the outcome of an oracle call at the midpoint. The search stops when $\mathcal{I}$ is small enough, according to some criterion.

Oracles can often be constructed from evaluators. For example, consider the one-parameter upper tolerance problem. Let $x^{(0)}$ be the optimum solution at $\mu^{(0)}$, and let u be the upper tolerance at $\mu^{(0)}$. Then, the parameter value being sought is $\mu^* = \mu^{(0)} + u$. The oracle must determine whether a given $\mu \geq \mu^{(0)}$ is less than or equal to μ^*. To test this, first use the evaluator to find an optimum solution $x^{(1)}$ at μ. If $cost(x^{(0)}, \mu) = \text{cost}(x^{(1)}, \mu)$, then $\mu \leq \mu^*$. Otherwise (since $x^{(1)}$ is optimum at μ), the only possibility is that $\text{cost}(x^{(0)}, \mu) < \text{cost}(x^{(1)}, \mu)$, and thus $\mu > \mu^*$.

Bisection search leads to fast algorithms for certain inverse sequence alignment problems with one or two parameters [20].

Megiddo's method. Megiddo's method [25,26] solves parametric search problems by simulating the execution of an evaluator for the underlying fixed-parameter problem to find its computation path at μ^*. This evaluator must be *piecewise linear*; that is, each value it computes must be a linear combination of the input parameters. This condition is not particularly restrictive, as many combinatorial optimization algorithms have this property. For example, most minimum spanning tree, maximum flow, and dynamic programming sequence alignment algorithms are piecewise linear. Like bisection search, Megiddo's method relies on repeated invocations of an oracle to narrow down the search range for μ^*. Indeed, the following result can be proved.

Theorem 30.1

Let $\mathcal{P}$ be a parametric search problem that has an oracle that runs in worst-case time b. Suppose that there exists a piecewise linear algorithm to evaluate $Z(\lambda)$ that executes t steps in the worst case. Then, $\mathcal{P}$ can be solved in time $O(t \cdot b)$. If the evaluator for $Z(\mu)$ is a piecewise linear parallel algorithm that has d parallel steps, using w processors, then $\mathcal{P}$ can be solved in time $O(d \cdot b \cdot \log w + d \cdot w)$.

The second of the above time bounds can be improved by a factor of $\log n$ for some problems [27]. Among these are several parametric search problems related to minimum spanning trees, including finding the maximizer of Z_G and the root of Z_G, when this function is strictly decreasing (as is the case for the minimum-ratio spanning tree problem). All these questions can be answered in $O(T_{MST}(n, m) \log n)$ time, where $T_{MST}(n, m)$ is the time to compute a minimum spanning tree in an n-vertex, m-edge graph. Megiddo's method also yields efficient algorithms for the minimum ratio cycle problem [27]. Theorem 30.1 can be extended to nonlinear problems and to any fixed number of parameters [28–31].

In some cases, such as parametric minimum spanning trees on planar or dense graphs or for certain optimization problems on graphs of bounded tree width, the polylogarithmic slowdown of Megiddo's technique relative to the fixed-parameter problem can be eliminated entirely [32–35]. This yields parametric algorithms that are asymptotically as fast as fixed-parameter evaluators.

Gradient ascent. *Gradient ascent* [14,36] can be used to search for the maximizer λ^* of the optimum cost function $Z(\lambda)$. The method is iterative, generating a sequence of points that converges to λ^*. If the current point $\lambda^{(0)}$ is not minimum, the algorithm chooses the next point by moving some distance (given by some predetermined increment sequence) in the direction of a *subgradient* of $Z(\lambda)$. Informally, a vector $s \in \mathbb{R}^d$ is a subgradient of Z at $\lambda^{(0)} \in \mathbb{R}^d$ if s points in a direction along which Z is nondecreasing; that is, s plays the role a gradient plays for a differentiable function. A subgradient of Z at $\lambda^{(0)}$ can be computed using an evaluator: If $x^{(0)}$ is an optimum solution at $\lambda^{(0)}$, then, the function $cost(x^{(0)}, \lambda)$ has the form $a_0 + \sum_{i=1}^{d} a_i\lambda_i$ and the vector $(a_1, \ldots, a_d)$ is a sub-gradient at $\lambda^{(0)}$. It can be shown that $\lambda^{(0)}$ is a maximizer of Z if the 0 vector is a subgradient at $\lambda^{(0)}$ [14]. In practice this termination condition can be hard to test, since only one subgradient is computed at any point. One way to handle this is by ending the search if the function has not decreased by a certain amount after some number of iterations. Although gradient ascent is fast in practice (indeed, it is widely used in Lagrangian relaxation), it is not in general possible to establish combinatorial bounds on its running time, since the convergence time of the algorithm depends on the choice of the increment sequence [14].

30.4.2 Construction Algorithms

Constructing the space decomposition $\mathcal{M}(Z)$ induced by the optimum cost function Z requires producing all the regions of $\mathcal{M}(Z)$, along with the incidence relationships between regions, and the optimum solution associated with each region. The construction problem can be solved by evaluating Z at various points in the parameter space, producing a sequence of hyperplanes, each of which is the cost function of some optimum solution. Each new hyperplane is used to incrementally update the current estimate of Z. The technique is reminiscent of the methods used to determine the shape of a convex polyhedron through a series of hyperplane probes [37,38] (see also Refs. [39,40]). Indeed, these results can be used to prove the following:

Theorem 30.2

Let d denote the number of parameters, let f and v be, respectively, the number of regions and vertices of $\mathcal{M}(Z)$, and let t be the time needed to evaluate Z. Then, $\mathcal{M}(Z)$ can be constructed in $O(t \cdot (f + dv))$ time, plus the time needed to construct the lower envelope of all the score functions generated during the computation. In particular, for $d = 1$ and 2, $\mathcal{M}(Z)$ can be constructed in time $O(t \cdot f)$ and $O(t \cdot f + f^2)$, respectively.

Lower envelope construction is dual to constructing the upper convex hull of a set of points in $\mathbb{R}^{d+1}$, a problem for which an extensive literature exists [41]. The time needed to construct a lower envelope depends heavily on the complexity of the output produced, as measured by its total number of faces of dimensions 0 through d. By the *Upper Bound Theorem* [42], the complexity of Z is $\Theta(f^{\lfloor (d+1)/2 \rfloor})$, where f is the number of regions. Thus, for $d \geq 3$, the time needed to build the lower envelope can dominate the total computation time.

An alternative method for building $\mathcal{M}(Z)$ is *lifting*, which works by executing the fixed-parameter problem for *all* parameter values simultaneously. Another construction technique relies on ray shooting (see Section 30.3). Applications of these techniques to parametric sequence alignment are given in Refs. [19,43,44].

30.4.3 Combinatorial Complexity Bounds for Small-Dimensional Problems

In some problems, the cost of a feasible solution is the weighted sum of a relatively small number of aggregate *features*. For example, in one common sequence alignment scoring scheme, while the number of feasible solutions (alignments) is exponentially large, the cost of a feasible solution depends only on the total number of matches, mismatches, and indels, each weighted by separate parameters Ref. [43]. In instances of this kind, the theorem below, from Ref. [45], is useful.

Theorem 30.3

Consider an instance of problem (30.2) where the cost of every feasible solution $x \in X$ can be expressed as $cost(x, \lambda) = \sum_{i=1}^{d} \lambda_i \cdot f_i(x)$, where $f_1(x), f_2(x), \ldots, f_d(x)$ are integers, called the features *of x. Suppose that there exist integers $n_1, \ldots, n_d$ such that for every $x \in X$ and each $i \in \{1, \ldots, d\}$, $0 \leq f_i(x) \leq n_1$. Then, $\mathcal{M}(Z)$ has $O\left(\left(\prod_{i=1}^{d} n_i\right)^{(d-1)/(d+1)}\right)$ regions.*

Two examples of the use of Theorem 30.3 are given next; applications to sequence alignment can be found in Ref. [43]. Note that Theorem 30.3 applies to certain NP-hard problems as well—for an example of this, see Ref. [46].

- An instance of the *stable marriage problem* has n men and n women. Each man m assigns a rank $r(m, w) \in [n]$ to every woman w, and each woman w assigns a rank $r(w, m) \in [n]$ to each man m.

A feasible solution is a pairing M of men and women called a *marriage*; M is *stable* if there is no man–woman pair $(m, w) \notin M$ such that m and w prefer each other to their current mate. The cost of M is $\lambda_1 x_M + \lambda_2 y_M$, where $x_M = \sum_{(m,w)\in M} r(m, w)$, $y_M = \sum_{(m,w)\in M} r(w, m)$, and λ_1 and λ_2 weigh the relative preference of men over women in forming M; x_M and y_M are the features of M. The goal is to find the stable marriage of minimum cost [47]. Since x_M and y_M are $O(n^2)$, Theorem 30.3 gives a bound of $O((n^2)^{2(1/3)}) = O(n^{4/3})$ on the number of regions of $\mathcal{M}(Z)$ (see also Ref. [48]).

- The input to the *ancestral reconstruction problem* [49] consists of an n-leaf-rooted binary tree T and a labeling $S_1, \ldots, S_n$ of T's leaves by DNA sequences of length k. A feasible solution (called an *ancestral reconstruction*) is a labeling of the internal nodes of T by DNA sequences of length k. Let $e = (x, y)$ be an edge of T whose endpoints are labeled by sequences X and Y and let $i \in [k]$ be an index for which $X[i] \neq Y[i]$. Then, there is a *transition* at i if either both $X[i]$ and $Y[i]$ are purines (A or G nucleotides) or both are pyrimidines (C or T nucleotides); otherwise, there is a *transversion* at i. The cost of a labeling L is $\lambda_1 v + \lambda_2 t$, where v and t are the total number of transversions and transitions over all the edges of T and all indices i; v and t are the features of L. The goal is to find a labeling of minimum cost. Since $0 \leq v, t \leq 2(n-1)k$, Theorem 30.3 implies that $\mathcal{M}(Z)$ has $O((nk)^{2(1/3)}) = O(n^{2/3}k^{2/3})$ regions.

In combination with Theorem 30.2, Theorem 30.3 allows us to bound the time needed to construct $\mathcal{M}(Z)$. For instance, the time to construct the optimum score function for the maximum parsimony problem is $O(n^{5/3}k^{5/3})$, since the number of regions is $O(n^{2/3}k^{2/3})$ and each evaluation takes $O(kn)$ time [49].

30.5 Sensitivity Analysis for Polynomially Solvable Problems

A number of sensitivity analysis algorithms are known for polynomially solvable problems. In addition to their intrinsic interest, these results are also useful in analyzing the stability of NP-hard problems. Some of this work has already been described in Section 30.4.1. The present section surveys problem-specific approaches, concentrating on linear programming, matroids, shortest paths, and network flows. In what follows, for graph problems, n and m denote the number of vertices and edges of the input graph, respectively.

30.5.1 Linear Programming

Parametric linear programming seems to have the distinction of being the first sensitivity analysis problem to be studied [1]. Consider first the simplest case, where the objective function depends on a single scalar parameter μ:

$$Z_{LP}(\mu) = \min\left\{\sum_{i=1}^{n}(c_i + \mu c_i')x_i : \sum_{i=1}^{n} a_{ij}x_i = b_j,\ \text{for } j \in \{1, \ldots, m\}, x_i \geq 0 \text{ for } i \in \{1, \ldots, n\}\right\} \tag{30.9}$$

If problem (30.9) is feasible, then it must have an optimum basic feasible solution (bfs) [50]. Since the number of such solutions is finite, function Z_{LP} is piecewise linear and concave. Its worst-case number of breakpoints is known to be exponential in n [51]. The upper and lower tolerances for an optimum bfs $x^{(0)}$ at $\mu^{(0)}$ can be computed using the *parametric simplex method* [50]. Parametric linear programming has also been studied from the perspective of interior point methods [52–54].

When the objective function depends on more than one parameter, the optimum cost function is again piecewise linear and concave; however, little is known about the complexity of the associated decomposition, other than that it must be exponential in the number of variables. Methods for multiparameter sensitivity analysis were first discussed in Ref. [55] (see also Refs. [56,57]).

By linear programming duality, all the results obtained for parametric linear programming apply to linear programming problems where the right-hand side of the constraint set is a function of a parameter vector [50].

30.5.2 Parametric Minimum Spanning Trees and Matroids

The minimum spanning tree problem is a special case of *matroid optimization* . A *matroid* [58] is a pair $M = (E, \mathcal{I})$, where E is a finite nonempty set of elements and $\mathcal{I}$ a family of subsets of E, called *independent sets*, satisfying two axioms: (i) any subset of an independent set is independent and (ii) if A and B are independent, with $|A| < |B|$, then there exists some $x \in B$ such that the set $A \cup \{x\}$ is independent. The *rank* of M is the cardinality of its largest independent set. A *base* of a matroid is a maximal independent set. A *weighted matroid* is one whose elements have real-valued weights. The *matroid optimization problem* is to find a minimum-weight base, in a weighted matroid; that is, a base minimizing the sum of the element weights.

The *graphic matroid* of an undirected graph G, denoted as $M(G)$, is a matroid whose elements are the edges of G, whose independent sets are the subforests of G, and whose bases are the spanning forests of G. Thus, when G is connected, a minimum-weight base in $M(G)$ is a minimum spanning tree of G, and the rank of $M(G)$ is $n - 1$. For further examples of matroids see Ref. [58].

Let $Z_M(\mu)$ be the function giving the weight of the optimum base of a matroid M where the weight of each element is a linear function of a scalar parameter μ. Then, $Z_M(\mu)$ has $O(mr^{1/3})$ breakpoints, where m and r are the number of elements and the rank of M, respectively [59]; this bound is tight in general [60]. The upper bound implies that the number of breakpoints for the parametric minimum spanning tree problem is $O(mn^{1/3})$. However, the best known lower bound for this special case is $\Omega(m\alpha(n))$, where α is the inverse Ackermann function [60].

The optimum cost function for the one-parameter minimum spanning tree problem can be constructed in time $O(n^{2/3} \log^{4/3} n)$ per breakpoint or in randomized expected time $O(n^{2/3} \log n)$, per breakpoint [61]. For planar graphs, the time bound can be improved to deterministic $O(n^{1/4} \log^{3/2} n)$ or randomized expected $O(n^{1/4} \log n)$ time per breakpoint. Further results for minor-closed families of graphs are given in Ref. [61]. The latter reference also offers algorithms for the *kinetic* minimum spanning tree problem, which extends the parametric minimum spanning tree problem by allowing arbitrary insertions or deletions of parametrically weighted edges.

All edge tolerances of a minimum spanning tree can be computed in deterministic $O(m \log \alpha(m, n))$ time [62] (see also Ref. [63]), where α is the inverse Ackerman function, and in randomized $O(m)$ time [64]. A deterministic $O(n)$-time algorithm exists for planar graphs [65]. These bounds match the bounds known for solving the minimum spanning tree problem with fixed-edge costs; indeed, it has been shown that any minimum spanning tree algorithm can be used to calculate minimum spanning tree edge tolerances, without an asymptotic loss in efficiency [62].

Algorithms for finding the most vital edges in minimum cost spanning trees can be found in Refs. [66–69] and in Chapter 62.

The robustness function for the minimum spanning tree problem is concave, piecewise linear, with $O(mn)$ breakpoints and can be constructed in $O(m^2 n^3 \log(n^2/m))$ time [17]. More generally, the robustness function of a weighted matroid can be constructed in $O(m^5 n^2 + m^4 n^4 \tau)$ time, where τ is the time needed to test the independence of a set of at most n elements [70].

When the number of parameters is fixed, the inverse parametric minimum spanning tree problem can be solved in randomized linear expected time, and deterministically in $O(m \log^2 n)$ worst-case running time. The inverse problem can be solved in polynomial time by means of the ellipsoid method for linear programming, even when the number of parameters is large [18]. This result extends to shortest path, matching, and other "optimal subgraph" problems.

30.5.3 Shortest Paths and Related Problems

Two kinds of shortest path sensitivity analysis problems have been considered in the literature. The first deals with the sensitivity of the shortest path tree rooted at a specified source vertex s. The edge tolerance problem for this case has been studied by several authors, often alongside the tolerance problem for minimum spanning trees [63,71,72]. Thus, all edge tolerances of a shortest path tree can be computed in deterministic $O(m \log \alpha(m, n))$ time [62]; for planar graphs, the same problem can be solved in $O(n)$ time [65]. Algorithms for the most vital edge problems on shortest path trees on undirected graphs have also been developed in Refs. [73,74] and Chapter 62.

The other kind of problem studies the sensitivity of the shortest path from a source s to a destination t. Ramaswamy et al. [75] have devised algorithms for two related questions. Let P^* be a shortest s–t path in an edge-weighted undirected graph G. Then, there is an $O(m + |P^*| \log |P^*|)$ algorithm for finding all upper and lower tolerances of all edges in G with respect to P^*. Ramaswamy et al. also consider the following max–min problem. Let G be an undirected graph where each edge has a real-valued capacity. The *capacity* of a path is the minimum capacity of an edge on the path. Let Q^* be the maximum capacity s–t path in G. Then, all upper and lower tolerances of the edges with respect to Q^* can be computed in $O(m + |Q^*| \log |Q^*|)$ time. These questions have applications to the pricing of edges in networks [76,77]. Note that, for the shortest path problem, there is a close relationship between computing tolerances and finding most vital edges. For instance, the most vital edge with respect to the shortest s–t path is the edge with the largest upper tolerance. In contrast, the most vital edge problem for minimum cost spanning trees is not directly related to the problem of computing edge tolerances.

Let $Z(\mu)$ be the weight of the shortest s–t path when each edge weight varies linearly as a function of a scalar parameter μ. Gusfield [78] shows that Z has $O(n^{\log n})$ breakpoints. Carstensen [79] provides evidence that this bound is tight.

30.5.4 Parametric Maximum Flow

The *maximum-flow problem* asks to find a source-to-sink flow of maximum value in a capacitated network G. A closely related question is the *minimum-cut problem*, which asks to find a partition (S, T) of the vertex set—that is, a *cut*—in G such that the source is in S and the sink is in T, and such that the total capacity of the edges from S to T is minimized. By the *max-flow min-cut theorem* [80], the value of the maximum flow in G equals that of the minimum cut in G.

In the parametric maximum-flow problem, the capacities are functions of a parameter vector. Not much is known about the properties of the general version of the problem, even for a single parameter; however, the following important special case has received considerable attention. A parametric flow network is *simple* if the capacity of each edge out of the source is a nondecreasing linear function of μ, the capacity of each edge into the sink is a nonincreasing function of μ, and the capacity of every other edge is constant; no arcs between source and sink are allowed. Simple parametric flow networks arise in diverse applications, including scheduling, stable marriage problems, finding maximum density subgraphs, and baseball elimination problems [81,82].

In simple networks the series of minimum cuts $(S_i, T_i), \ldots, (S_r, T_r)$ encountered as μ is increased have a *nesting property*; that is, $S_i \subset S_{i+1}$, for $i = 1, \ldots, r-1$ [39]. Thus, the function $Z_C(\mu)$ giving the capacity of the minimum cut has at most $n-2$ breakpoints. Remarkably, Gallo et al. [81] show that the preflow-push algorithm [83] can be modified to construct $\mathcal{M}(Z_C)$ in $O(nm \log(n^2/m)$ time, the same time as that algorithm takes to compute a *single* maximum flow. They also show how to achieve the same running time for finding the value of μ at which $Z_C(\mu)$ is 0, as required in certain minimum-ratio applications, and for locating a maximizer of $Z_C(\mu)$. Further applications and extensions are given in Ref. [82]; improvements for bipartite flow networks are given in Ref. [84]. Instances with piecewise-linear capacities or where certain arcs not incident on s or t have varying capacities—which have applications to preemptive scheduling with release times and deadlines, where processing times are varied, at a cost—have also been studied [85].

30.6 Sensitivity Analysis of NP-Hard Problems

This section gives an overview of the work on sensitivity analysis for NP-hard problems, primarily with respect to changes in the coefficient vector.

30.6.1 Complexity of Sensitivity Analysis

Unsurprisingly, several results exist indicating that analyzing the sensitivity of NP-hard problems is itself hard. Thus, Hall and Posner [86] have shown that, for a large class of NP-hard scheduling problems, performing sensitivity analysis relative to the processing times is itself NP-hard. Their results (which actually apply to the problem of recomputing the optimum solution after a specific change of parameters) strongly suggest that the computation of tolerances, stability regions, and radii is hard.

van Hoesel and Wagelmans [87] have proved a variety of hardness results for 0/1 problems with linear objective function, including the following:

- If the tolerances can be computed in polynomial time, then (under some mild assumptions) the fixed-parameter version of the problem can be solved in polynomial time as well. Thus, for instance, there is no polynomial-time algorithm for determining the edge tolerances for the traveling salesman problem unless P = NP.
- If it can be checked in polynomial time whether an optimum solution $x^{(0)}$ for a given cost vector $c^{(0)}$ is also optimum for another cost vector $c^{(1)}$, then the upper and lower tolerances at $c^{(0)}$ can be computed in polynomial time. Thus, it is NP-hard to test whether the optimum solution to an NP-hard 0/1 optimization problem remains optimum after arbitrary parameter changes.
- If there is a polynomial-time ϵ-approximate algorithm for computing all the upper tolerances for a 0/1 problem, then there exists a polynomial-time algorithm to compute the upper tolerances exactly. Thus, it is NP-hard to compute the upper tolerances of the optimum solution to an NP-hard 0/1 optimization problem approximately.
- If all upper tolerances for an ϵ-approximate solution $x^{(0)}$ at $c^{(0)}$ can be computed in polynomial time, then the optimum solution at $c^{(0)}$ can be computed in polynomial time. Thus, for instance, the existence of a polynomial algorithm to determine upper tolerances for an ϵ-approximate solution to the minimum-cost knapsack problem, would imply that P = NP.

From the above results, one can conclude that computing stability regions and radii for ϵ-approximate solutions are hard problems in general.

Sotskov et al. [21] establish a relationship between the complexity of computing exact solutions and that of multiparameter sensitivity analysis. Let x be an ϵ-optimal solution and let $\nu(x)$ denote the number of 1's in x. Let $c^{(0)}$ be a coefficient vector and $x^{(0)}$ an ϵ-approximate solution at $c^{(0)}$. Then, if the optimum cost $Z(c)$ can be calculated in $O(g(n))$ time for any coefficient vector $c \in \mathbb{R}^n$, the stability radius of x at $c^{(0)}$ can be computed in $O(2^{\nu(x)}ng(n))$ time. In contrast, Chakravarti and Wagelmans [88] show that the stability radius can be computed in polynomial time for problems with linear objective functions whose fixed-parameter versions can be solved in polynomial time.

While the preceding discussion was limited to problems with linear objective function, it should be noted that several results are known for bottleneck problems. See, for example, Refs. [89–91]; the last of these references surveys the extensive treatment of the subject in the Russian literature.

30.6.1.1 Graphs of Bounded Tree Width

Many NP-complete graph optimization problems are polynomially solvable on graphs of constant-bounded tree width [92–95] (for a definition of tree width, see Refs. [96,97]). Indeed, parametric versions of several of these problems, where edge and/or vertex costs are linear functions of a single scalar parameter μ, are often themselves efficiently solvable [34,98]. This has some implications to the approximate parametric analysis of NP-hard problems, through Baker's [99] technique for obtaining polynomial-time approximation schemes for certain (nonparametric) problems on weighted planar graphs. Among these problems are maximum independent set, maximum tile salvage, partition into triangles, maximum H-matching, minimum vertex cover, minimum dominating set, and minimum edge dominating set.

Baker's approach is based on decomposing the input planar graph into k-*outerplanar* graphs, which are graphs of constant-bounded tree width. Thus, one can use parametric algorithms for bounded-tree width graphs to obtain approximation algorithms for constructing the optimum cost function for all of

Baker's problems. More precisely, let $Z(\mu)$ denote the optimal cost function for one of Baker's problems. Then, for every fixed ϵ, there exists a polynomial-time algorithm that produces a function $Z_\epsilon(\mu)$ that has polynomially many breakpoints and such that for each μ, $Z_\epsilon(\mu)$ is ϵ-approximate to $Z(\mu)$ [34].

30.6.2 Bounding the Stability Region

The notions of *restriction* and *relaxation* can help in obtaining bounds on the stability region. Let $cost_A$ and X_A denote the cost function and set of feasible solutions of a problem A of the form (30.1). Problem Q is a *restriction* of problem P if $X_Q \subseteq X_P$ and $cost_Q(x) \geq cost_P(x)$ for every $x \in X_Q$. Problem R is a *relaxation* of problem P if $X_R \supseteq X_P$ and $cost_R(x) \leq cost_P(x)$ for every $x \in X_P$. Note that restriction and relaxation are inverse relations; that is, R is a relaxation of P if and only if P is a restriction of R. Also, the cost of the optimum solution of any relaxation (restriction) of P is a lower (upper) bound on the cost of the optimum solution for P. These facts lead to the following elementary, but useful, observations [4]:

(GN1) If an optimum solution x^* of P is feasible for a restriction Q of P and $cost_Q(x^*) = cost_P(x^*)$, then x^* is optimum for Q.

(GN2) If $x^{(0)}$ is a feasible solution for P and $cost_P(x^{(0)})$ equals the cost of an optimum solution for some relaxation of P, then $x^{(0)}$ must be optimum for P.

Observation (GN1) allows one to make statements about the sensitivity of an optimum solution x^* to changes in the coefficient vector. For example, x^* remains optimum if c is replaced by any c' such that $c'_j \geq c_j$ for every j such that $x^*_j = 0$ and $c'_j = c_j$ otherwise; that is, each such c_j has infinite upper tolerance. Similarly, for a 0/1 optimization problem, x^* remains optimum if c is replaced by any c' such that $c'_j \leq c_j$ for every j such that $x_j = 1$ and $c'_j = c_j$ otherwise; that is, each such c_j has infinite lower tolerance.

Hall and Posner [86] note that observation (GN1) can be used to study the sensitivity of scheduling problems. As a simple example, an optimal schedule remains optimal if the release date of a job is increased, but not beyond the time when the job starts processing.

Observation (GN2) applies to Lagrangian relaxation (Section 30.2): Suppose that λ^* is the maximizer of the Lagrangian function (Eq. [30.4]), and that x^* is optimum solution for this parametric problem at λ^*. Then, if x^* is also feasible for the original problem (30.3), x^* must be optimum for the latter problem. Thus, one can estimate the stability of x^* in the original (hard) problem, by analyzing the sensitivity of x^* in the relaxed (usually polynomially solvable) problem, relying on results such as those presented in Section 30.5. Along these lines, Libura [100] presents methods for computing lower bounds on the edge tolerances for the traveling salesman problem and the minimum-weight Hamiltonian path problems, which rely on relaxations to minimum spanning tree problems. This allows one to use the results on sensitivity analysis for minimum spanning trees described in Section 30.5.2.

Sotskov et al. [21] study the stability region of ϵ-optimal solutions and prove certain necessary and sufficient conditions to obtain lower and upper bounds on the stability radius. They considered the stability of non-preemptive general shop scheduling with precedence constraints, when the processing times of the jobs are varied and provide conditions for the stability radius to be strictly larger than 0 or for it to be infinite. Unfortunately, verifying these conditions is not easy.

30.6.2.1 *k* Best Solutions and Sensitivity Analysis

Intuitively, the boundaries of the stability region of an optimum $x^{(0)}$ at $\lambda^{(0)}$ are determined by a set of near-optimum solutions at $\lambda^{(0)}$. This observation motivated Libura et al. [101] and van der Poort [102] to study the connections between near optimality and sensitivity analysis for the traveling salesman problem. Unsurprisingly, they show that it is NP-hard to find the smallest k such that the k best solutions at $c^{(0)}$ suffice to determine the stability region of an optimum tour at $c^{(0)}$. Nevertheless, they do obtain some positive results, which are reviewed below.

Suppose one is given a set S containing the k best tours for some edge cost vector $c^{(0)}$; S of course includes an optimal tour, $H^{(0)}$. Let E_1 be the set of edges that are present in $H^{(0)}$, but not in every tour in S. Then, the upper tolerances of the costs of the edges in E_1 can be computed exactly and one can obtain lower bounds on the upper tolerances of the other edges. Similarly, let E_2 be the set of edges present in some tour in S. Then, one can calculate the lower tolerances of the edges present in E_2 and lower bounds on the lower tolerances of the other edges. The authors also show how to derive upper and lower bounds for the stability radius and how to use this information to derive subsets of the stability region. The authors evaluated their results experimentally on instances from TSPLIB [103]. They observe that the number of edges for which the tolerances can be computed increases quickly for smaller values of k, but that the growth rate decreases for larger values of k.

30.6.3 Sensitivity of Heuristics

Analyzing the stability of a heuristic algorithm is conceivably easier than analyzing the stability of an exact or ϵ-approximate solution, since the heuristic need not have any sort of performance guarantee (recall the distinction between algorithm and solution stability, explained at the end of Section 30.3). For example, Ghosh et al. [104] consider the greedy heuristics for binary knapsack and subset-sum problems. They show that the upper and lower tolerances of this algorithm for any parameter (the knapsack capacity, the weights, and the profits of the items) can be computed in polynomial time. They also study the conditions under which the sensitivity analysis of the heuristic generates bounds for the tolerances for the optimal solutions, and the empirical behavior of the greedy output when there is a change in the problem data. Hall and Posner [86] study the behavior of greedy heuristics for two NP-hard problems: scheduling two machines to minimize the weighted completion time and scheduling two machines to minimize the makespan. They provide approaches to obtain bounds on the upper tolerances of each heuristic to changes in the processing times.

Intuitively, since a simple heuristic uses less of the input information than an exact algorithm, it may produce a poor result, but the solution should be less susceptible to parameter changes than the optimal solution. This intuition is supported by the work of Kolen et al. [105] comparing two heuristics for scheduling jobs on identical parallel machines to minimize the makespan: *Shortest Processing Time* (SPT) and *Longest Processing Time* (LPT). The ratios between the solution returned and the optimum are $2 - 1/m$ for SPT, where m is the number of machines and $4/3 - 1/(3m)$ for LPT [106,107].

Suppose the processing time μ of one of the jobs is varied from 0 to ∞. For $H \in \{SPT, LPT\}$, let $Z_H(\mu)$ be the length of the makespan of the schedule returned by H, let $B_H(n, m)$ denote the worst-case number of breakpoints of Z_H, and let $A_H(n, m)$ be the number of different assignments of jobs to machines. The latter two values serve as an indication of sensitivity of H. Kolen et al. show that $Z_H(\mu)$ is a continuous piecewise linear function. They also prove that $A_{SPT}(n, m) \leq n$ and $B_{SPT}(n, m) \leq 2\lceil n/m \rceil$ and that both bounds are tight. In contrast, $A_{LPT}(n, m) \leq 2^{n-m}$ and $B_{LPT}(n, m) \leq 2^{n-m+1}$. The first bound is tight and there exists an example for which $B_{LPT}(n, m) > 2^{(n-m)/2}$. These results support the intuition that the sensitivity of a list scheduling rule increases with the quality of the schedule produced.

Acknowledgment

Partially supported by grants CCR-9988348 and EF-0334832 from the National Science Foundation.

References

[1] Gass, S. I. and Saaty, T., The computational algorithm for the parametric objective function, *Naval Res. Logistics Quart.*, 2, 39, 1955.

[2] Hoffman, H. and Jacobs, W., Smooth patterns of production, *Manage. Sci.*, 1, 86, 1954.

[3] Gal, T. and Greenberg, H. J., Eds., *Advances in Sensitivity Analysis and Parametric Programming*, International Series in Operations Research and Management Science, Kluwer, Dordrecht, 1997.

[4] Geoffrion, A. M. and Nauss, R., Parametric and postoptimality analysis in integer linear programming, *Manage. Sci.*, 23(5), 453, 1977.

[5] Wagelmans, A., Sensitivity analysis in combinatorial optimization, Ph.D. thesis, Econometric Institute, Erasmus University, Rotterdam, The Netherlands, 1990.

[6] Greenberg, H., An annotated bibliography for post-solution analysis in mixed integer programming and combinatorial optimization, in *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search*, Woodruff, D., Ed., Kluwer, Dordrecht, 1998, p. 97. (Note: updated bibliography at `http://www.cudenver.edu/hgreenbe/aboutme/papers/mipbib.pdf`.)

[7] Downey, R. G. and Fellows, M. R., *Parameterized Complexity*, Springer, New York, 1999.

[8] Hromkovič, J., Stability of approximation algorithms for hard optimization problems, in *SOFSEM*, Lecture Notes in Computer Science, Vol. 1725, Springer, Berlin, 1999, p. 29.

[9] Bender, M. A. and Chekuri, C., Performance guarantees for the TSP with a parameterized triangle inequality, *Inf. Process. Lett.*, 73(1–2), 17, 2000.

[10] Held, M. and Karp, R., The traveling salesman problem and minimum spanning trees, *Oper. Res.*, 18, 1138, 1970.

[11] Held, M. and Karp, R., The traveling salesman problem and minimum spanning trees: part II, *Math. Prog.*, 6, 6, 1971.

[12] Garey, M. R. and Johnson, D. S., *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[13] Fisher, M. L., The Lagrangian relaxation method for solving integer programming problems, *Manage. Sci.*, 27(1), 1, 1981.

[14] Nemhauser, G. L. and Wolsey, L. A., *Integer and Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematic and Optimization, Wiley, New York, 1988.

[15] Chandrasekaran, R., Minimal ratio spanning trees, *Networks*, 7, 335, 1977.

[16] Lawler, E. L., Optimal cycles in doubly-weighted linear graphs, *Proc. Int. Symp. on Theory of Graphs,* 1966, p. 209.

[17] Frederickson, G. N. and Solis-Oba, R., Increasing the weight of minimum spanning trees, *J. Algorithms*, 33, 394, 1999.

[18] Eppstein, D., Setting parameters by example, *SIAM J. Comput.*, 32(3), 643, 2003.

[19] Gusfield, D. and Stelling, P., Parametric and inverse-parametric sequence alignment with XPARAL, in *Computer Methods for Macromolecular Sequence Analysis*, Doolittle, R. F., Ed., Methods in Enzymology, Vol. 266, Academic Press, New York, 1996, p. 481.

[20] Sun, F., Fernández-Baca, D., and Yu, W., Inverse parametric sequence alignment, *J. Algorithms*, 53(1), 36, 2004.

[21] Sotskov, Y. N., Wagelmans, A. P. M., and Werner, F., On the calculation of the stability radius of an optimal or an approximate schedule, *Ann. Oper. Res.*, 83, 213, 1998.

[22] Radzik, T., Newton's method for fractional combinatorial optimization, *Proc. of FOCS,* 1992, p. 659.

[23] Radzik, T., Parametric flows, weighted means of cuts, and fractional combinatorial optimization, in *Complexity in Numerical Computations*, Pardalos, P., Ed., World Scientific, Singapore, 1993, p. 351.

[24] Radzik, T., *Algorithms for some linear and fractional combinatorial optimization problems*, Department of Computer Science, Ph.D. Thesis, Stanford University, 1992.

[25] Megiddo, N., Combinatorial optimization with rational objective functions, *Math. Oper. Res.*, 4, 414, 1979.

[26] Megiddo, N., Applying parallel computation algorithms in the design of serial algorithms, *JACM*, 30(4), 852, 1983.

[27] Cole, R., Slowing down sorting networks to obtain faster sorting algorithms, *JACM*, 34(1), 200, 1987.

[28] Agarwala, R. and Fernández-Baca, D., Weighted multidimensional search and its application to convex optimization, *SIAM J. Comput.*, 25, 83, 1996.

[29] Cohen, E. and Megiddo, N., Maximizing concave functions in fixed dimension, in *Complexity in Numerical Optimization*, Pardalos, P. M., Ed., World Scientific, Singapore, 1993, p. 74.

[30] Fernández-Baca, D., On non-linear parametric search, *Algorithmica*, 30, 1, 2001.

[31] Toledo, S., Maximizing non-linear concave functions in fixed dimension, in *Complexity in Numerical Computations*, Pardalos, P., Ed., World Scientific, Singapore, 1993, p. 429.

[32] Frederickson, G. N., Optimal algorithms for tree partitioning, *Proc. of SODA,* 1991, p. 168.

[33] Fernández-Baca, D., Slutzki, G., and Eppstein, D., Using sparsification for parametric minimum spanning tree problems, *Nordic J. Comput.*, 34(4), 352, 1996.

[34] Fernández-Baca, D. and Slutzki, G., Optimal parametric search on graphs of bounded tree-width, *J. Algorithms*, 22, 212, 1997.

[35] Fernández-Baca, D. and Slutzki, G., Linear-time algorithms for parametric minimum spanning tree problems on planar graphs, *Theor. Comput. Sci.*, 181, 57, 1997.

[36] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., *Numerical Recipes: The Art of Scientific Computing*, 2nd ed., Cambridge University Press, Cambridge, UK and New York, 1992.

[37] Dobkin, D., Edelsbrunner, H., and Yap, C. K., Probing convex polytopes, *Proc. of STOC,* 1986, p. 424.

[38] Dobkin, D., Edelsbrunner, H., and Yap, C. K., Probing convex polytopes, in *Autonomous Robot Vehicles*, Cox and Wilfong, Eds., Springer-Verlag, NY, 1990, p. 328.

[39] Eisner, M. and Severance, D., Mathematical techniques for efficient record segmentation in large shared databases, *JACM*, 23, 619, 1976.

[40] Fernández-Baca, D. and Srinivasan, S., Constructing the minimization diagram of a two-parameter problem, *Oper. Res. Lett.*, 10, 87, 1991.

[41] Seidel, R., Convex hull computations, in *Handbook of Discrete and Computational Geometry*, Goodman, J. and O'Rourke, J., Eds., CRC Press LLC, Boca Raton, FL, 1997, chap. 19.

[42] McMullen, P. The maximum number of faces of a convex polytope, *Mathematika*, 17, 179, 1970.

[43] Fernández-Baca, D. and Venkatachalam, B., Parametric sequence alignment, in *Handbook of Computational Molecular Biology*, Aluru, S., Ed., Chapman & Hall/CRC, Boca Raton, FL, 2005.

[44] Pachter, L. and Sturmfels, B., Parametric inference for biological sequence analysis, *Proc. Natl. Acad. Sci. USA*, 101(46), 2004, 16138.

[45] Pachter, L. and Sturmfels, B., Tropical geometry of statistical models, *Proc. Natl. Acad. Sci. USA*, 101(46), 2004, 16132.

[46] Fernández-Baca, D., Seppäläinen, T., and Slutzki, G., Parametric multiple sequence alignment and phylogeny construction, *J. Discrete Algorithms*, 2, 271, 2004.

[47] Gusfield, D. and Irving, R. W., *The Stable Marriage Problem: Structure and Algorithms*, MIT Press, Cambridge, MA, 1989.

[48] Gusfield, D. and Irving, R. W., Parametric stable marriage and minimum cuts, *Inf. Process. Lett.*, 30, 255, 1989.

[49] Felsenstein, J., *Inferring Phylogenies*, Sinauer Assoc., Sunderland, MA, 2003.

[50] Vanderbei, R. J., *Linear Programming: Foundations and Extensions*, *International Series in Operations Research and Management Science*, Vol. 37, Springer-Verlag, NY, 2001.

[51] Murty, K., Computational complexity of parametric linear programming, *Math. Prog.*, 19, 213, 1980.

[52] Adler, I. and Monteiro, R. D. C., A geometric view of parametric linear programming, *Algorithmica*, 8(2), 161, 1992.

[53] Berkelaar, A., Roos, C., and Terlaky, T., The optimal set and optimal partition approach to linear and quadratic programming, in *Recent Advances in Sensitivity Analysis and Parametric Programming*, Greenberg, H. and Gal, T., eds., Kluwer, Dordrecht, 1997, chap. 6.

[54] Yildirim, E. A. and Todd, M. J., Sensitivity analysis in linear programming and semidefinite programming using interior-point methods, *Math. Prog.*, 90(2), 229, 2001.

[55] Gal, T. and Nedoma, J., Multiparametric linear programming, *Manage. Sci.*, 18, 406, 1972.

[56] Borrelli, F., Bemporad, A., and Morari, M., Geometric algorithm for multiparametric linear programming, *J. Opt. Theor. Appl.*, 118(3), 515, 2003.

[57] Schechter, M., Polyhedral functions and multiparametric linear programming, *J. Opt. Theor. Appl.*, 53, 269, 1987.

[58] Welsh, D. J. A., *Matroid Theory*, Academic Press, New York, 1976.

[59] Dey, T., Improved bounds on planar k-sets and related problems, *Discrete and Comput. Geom.*, 19(3), 373, 1998.

[60] Eppstein, D., Geometric lower bounds for parametric matroid optimization, *Discrete Comput. Geom.*, 20, 463, 1998.

[61] Agarwal, P. K., Eppstein, D., Guibas, L. J., and Henzinger, M. R., Parametric and kinetic minimum spanning trees, *Proc. of FOCS*, 1998.

[62] Pettie, S., Sensitivity analysis of minimum spanning trees in sub-inverse-Ackermann time, *Proc. of ISAAC*, Lecture Notes in Computer Science, Vol. 3827, Springer, Berlin, 2005, p. 964.

[63] Tarjan, R. E., Sensitivity analysis of minimum spanning trees and shortest path problems, *Inf. Process. Lett.*, 14(1), 30, 1982.

[64] Dixon, B., Rauch, M., and Tarjan, R., Verification and sensitivity analysis of minimum spanning trees in linear time, *SIAM J. Comput.*, 21, 1184, 1994.

[65] Booth, H. and Westbrook, J., A linear algorithm for analysis of minimum spanning and shortest-path trees of planar graphs, *Algorithmica*, 11, 341, 1994.

[66] Hsu, L. H., Jan, R., Lee, Y., Hung, C., and Chern, M., Finding the most vital edge with respect to minimum spanning tree in weighted graphs, *Inf. Process. Lett.*, 39, 277, 1991.

[67] Hsu, L., Wang, P., and Wu, C., Parallel algorithms for finding the most vital edge with respect to minimum spanning tree, *Parallel Comput.*, 18, 1143, 1992.

[68] Iwano, K. and Katoh, N., Efficient algorithms for finding the most vital edge of a minimum spanning tree, *Inf. Process. Lett.*, 48, 211, 1993.

[69] Banerjee, S. and Saxena, S., Parallel algorithm for finding the most vital edge in weighted graphs, *J. Parallel Distr. Comput.*, 46, 101, 1997.

[70] Frederickson, G. N. and Solis-Oba, R., Algorithms for measuring perturbability in matroid optimization, *Combinatorica*, 18(4), 503, 1998.

[71] Shier, D. and Witzgall, C., Arc tolerances in shortest path and network flow problems, *Networks*, 10(4), 277, 1980.

[72] Gusfield, D., A note on arc tolerances in sparse shortest-path and network flow problems, *Networks*, 13, 191, 1983.

[73] Malik, K., Mittal, A., and Gupta, S., The k most vital edges in the shortest path problem, *Oper. Res. Lett.*, 8, 223, 1989.

[74] Venema, S., Shen, H., and Suraweera, F., NC algorithms for the single most vital edge problem with respect to shortest paths, *Inf. Process. Lett.*, 60, 243, 1996.

[75] Ramaswamy, R., Orlin, J. B., and Chakravarti, N., Sensitivity analysis for shortest path problems and maximum capacity path problems in undirected graphs, *Math. Prog.*, 102(2, Ser. A.), 355, 2005.

[76] Hershberger, J. and Suri, S., Vickrey prices and shortest paths: what is an edge worth? *Proc. of FOCS*, 2001, p. 252.

[77] Hershberger, J. and Suri, S., Erratum to: Vickrey prices and shortest paths: what is an edge worth? *Proc. of FOCS*, 2002, p. 809.

[78] Gusfield, D., Sensitivity Analysis for Combinatorial Optimization, TR UCB/ERL M80/22, University of California, Berkeley, 1980.

[79] Carstensen, P., Complexity of some parametric integer and network programming problems, *Math. Prog.*, 26, 64, 1983.

[80] Ahuja, R., Magnanti, T., and Orlin, J., *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[81] Gallo, G., Grigoriades, M., and Tarjan, R., A fast parametric maximum flow algorithm and applications, *SIAM J. Comput.*, 18, 30, 1989.

[82] Gusfield, D. and Martel, C., A fast algorithm for the generalized parametric minimum cut problem and applications, *Algorithmica*, 7, 499, 1992.

[83] Goldberg, A. V. and Tarjan, R. E., A new approach to the maximum-flow problem, *JACM*, 35(4), 921, 1988.

[84] Ahuja, R., Orlin, J., Stein, C., and Tarjan, R., Improved algorithms for bipartite network flow, *SIAM J. Comput.*, 23(5), 906, 1994.

[85] McCormick, S. T., Fast algorithms for parametric scheduling come from extensions to parametric maximum flow, *Oper. Res.*, 47(5), 744, 1999.

[86] Hall, N. G. and Posner, M. E., Sensitivity analysis for scheduling problems, *J. Scheduling*, 7(1), 49, 2004.

[87] van Hoesel, S. and Wagelmans, A., On the complexity of postoptimality analysis of 0/1 programs, *Discrete Appl. Math.*, 91(1–3), 251, 1999.

[88] Chakravarti, N. and Wagelmans, A. P.M., Calculation of stability radii for combinatorial optimization problems, *Oper. Res. Lett.*, 23(1–2), 1, 1998.

[89] Ramaswamy, R. and Chakravarti, N., Complexity of Determining Exact Tolerances for Min–Sum and Min–Max Combinatorial Optimization Problems, Working paper WPS-247/95, Indian Institute of Management, Calcutta, India, 1995.

[90] Ramaswamy, R., Chakravarti, N., and Ghosh, D., Complexity of Determining Exact Tolerances for Min–Max Combinatorial Optimization Problems, SOM report 00A22, University of Groningen, 2000.

[91] Sotskov, Y. N., Leontev, V. K., and Gordeev, E. N., Some concepts of stability analysis in combinatorial optimization, *Discrete Appl. Math.*, 58(2), 169, 1995.

[92] Arnborg, S. and Proskurowski, A., Linear time algorithms for NP-hard problems restricted to partial *k*-trees, *Discrete Appl. Math.*, 23(1), 11, 1989.

[93] Bern, M. W., Lawler, E. L., and Wong, A. L., Linear time computation of optimal subgraphs of decomposable graphs, *J. Algorithms*, 8, 216, 1987.

[94] Bodlaender, H. L., Treewidth: algorithmic techniques and results, *Proc. Int. Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 1295, Springer, Berlin, 1997, p. 29.

[95] Courcelle, B., Graph rewriting: an algebraic and logic approach, in *Handbook of Theoretical Computer Science*, Vol. B, van Leeuwen, J., Ed., Elsevier, Amsterdam, 1990, chap. 5.

[96] Robertson, N. and Seymour, P. D., Graph minors II: algorithmic aspects of tree-width, *J. Algorithms*, 7, 309, 1986.

[97] Diestel, R., Graph theory, *Graduate Texts in Mathematics,* Vol. 173, Springer, Berlin, 2005.

[98] Fernández-Baca, D., and Slutzki, G., Parametric problems on graphs of bounded tree-width, *J. Algorithms*, 16, 408, 1994.

[99] Baker, B. S., Approximation algorithms for NP-complete problems on planar graphs, *JACM*, 41(1), 153, 1994.

[100] Libura, M., Sensitivity analysis for minimum Hamiltonian path and traveling salesman problems, *Discrete Appl. Math.*, 30(2–3), 197, 1991.

[101] Libura, M., van der Poort, E. S., Sierksma, G., and van der Veen, J. A. A., Stability aspects of the traveling salesman problem based on *k*-best solutions, *Discrete Appl. Math.*, 87(1–3), 159, 1998.

[102] van der Poort, E. S., Aspects of Sensitivity Analysis for the Traveling Salesman Problem, Ph.D. thesis, University of Groningen, 1997.

[103] Reinelt, G. and Bixby, B., TSPLIB, `http://nhse.cs.rice.edu/softlib/catalog/tsplib.html`.

[104] Ghosh, D., Chakravarti, N., and Sierksma, G., Sensitivity analysis of the greedy heuristic for binary knapsack problems, *Eur. J. Oper. Res.*, 169(2), 340, 2006.

[105] Kolen, A. W. J., Rinnooy Kan, A. H. G., van Hoesel, C. P. M., and Wagelmans, A. P. M., Sensitivity analysis of list scheduling heuristics, *Discrete Appl. Math.*, 55(2), 145, 1994.

[106] Graham, R. L., Bounds for certain multiprocessing anomalies, *Bell Syst. Tech. J.*, 45, 1563, 1966.

[107] Graham, R. L., Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.*, 17(2), 416, 1969.

31 Stability of Approximation

Hans-Joachim Böckenhauer
Swiss Federal Institute of Technology (ETH) Zurich

Juraj Hromkovič
Swiss Federal Institute of Technology (ETH) Zurich

Sebastian Seibert
Swiss Federal Institute of Technology (ETH) Zurich

31.1 Introduction

The design of approximation algorithms has evolved as one of the most successful approaches for dealing with hard optimization problems. The first approximation algorithms were introduced in the 1960s [1,2]. A short time after introducing NP-hardness (and completeness) as a concept for proving the intractability of computational problems [3,4], a large range of successful approximation algorithms was proposed [5–10]. Since then, a wealth of approximation algorithms for very many practically relevant problems have been developed, as large parts of this handbook stand witness to. Also, overviews on approximation algorithms can be found, for example, in Refs. [11–14] and in Chapter 1 of this handbook.

Approximation algorithms allow us to jump from exponential time complexity (unless P=NP) to polynomial time complexity by a small change in the requirement from searching for the optimal solution to searching for a near-optimal solution. This effect is especially strong in the case of so-called approximation schemes, where one can guarantee to find a solution whose cost can become arbitrarily close to the optimal value (see Chapter 9 for more details).

Another possibility to make intractable problems tractable is to consider only a subset of input instances satisfying some special properties instead of the set of all inputs for which the problem is well defined. A good example is the Traveling Salesman Problem (TSP) which has served from the beginning as an archetype of a hard problem [15,16]. The traveling salesman problem is not only NP-hard, but also the search for an α-approximate solution for TSP is NP-hard for every constant α [6]. But if one considers TSP for inputs satisfying the triangle inequality (the so-called Δ-TSP), one can even design an approximation algorithm [10] with approximation ratio $\alpha = \frac{3}{2}$. The situation is still more interesting, if one considers the Euclidean TSP, where the vertices are points in the Euclidean space and the edge weights correspond to the distances between the vertices in the Euclidean metrics. The Euclidean TSP is strongly NP-hard [17], but for every small $\varepsilon > 0$ one can design a polynomial-time $(1 + \varepsilon)$-approximation algorithm [18–20] (or polynomial-time approximation scheme [PTAS]).

The fascinating observations of huge quantitative changes as mentioned above have lead to the proposal of considering the "stability" of approximation algorithms. Let us consider the following scenario. One has an optimization problem P for two sets of inputs L_1 and L_2, $L_1 \subset L_2$. For L_1 there exists a polynomial-time α-approximation algorithm A, but for L_2 there is no polynomial-time δ-approximation algorithm for any $\delta > 1$ (unless NP=P). We pose the following question: Is the algorithm A really useful for inputs from L_1 only? Let us consider a metrics M in L_2 determining the distance between any two inputs in L_2. Now, one can consider an input $x \in L_2 - L_1$, for which there exists an $y \in L_1$ such that $distance_M(x, y) \leq k$ for some positive real number k. One can look for how "good" the algorithm A is for the input $x \in L_2 - L_1$. If, for every $k > 0$ and every x with distance at most k to L_1, A computes a $\delta_{\alpha,k}$-approximation of an optimal solution for x (where $\delta_{\alpha,k}$ is considered to be a constant depending on k and α only), then one can say that A is "(approximation) stable" according to the metrics M.

The idea of this concept is similar to the concept of stability of numerical algorithms. But instead of observing the size of the change of the output value according to a small change of the input value, we look for the size of the change of the approximation ratio according to a small change in the specification (some parameters, characteristics) of the set of problem instances considered. If the change of the approximation ratio is small for every small change in the specification of the set of problem instances, then we have a stable algorithm. If a small change in the specification of the set of problem instances causes an essential (depending on the size of the input instances) increase of the relative error, then the algorithm is unstable.

The concept of stability enables us to show positive results extending the applicability of known approximation algorithms. As we shall see later, the concept also motivates to modify an unstable algorithm A to get a stable algorithm B that achieves the same approximation ratio on the original set of problem instances as A has, but can also be successfully used outside of this original set. This concept is useful because there are a number of problems for which an additional assumption on the "parameters" of the problem instances leads to an essential decrease of the hardness of the problem. Such effects are the starting points for trying to partition the whole set of problem instances into a spectrum of classes according to their polynomial-time approximability.

As one can observe, this approach is similar to the concept of parameterized complexity, introduced by Downey and Fellows [21,22], in trying to overcome the troubles caused by measuring complexity and approximation ratio in the worst-case manner. The main aim of both concepts is in partitioning the set of all instances of a hard problem into infinitely many classes with respect to the hardness of particular instances. We believe that approaches like these will be the core of future algorithmics, because they provide a deeper insight into the nature of the hardness of specific problems. In many applications we are not interested in the worst-case problem hardness, but in the hardness of forthcoming problem instances.

This chapter is organized as follows. In Section 31.2, we formally define the concept of approximation stability. In Section 31.3, we apply the concept of stability to the TSP and exhibit a partition of the general TSP input instances into infinitely many classes, according to their approximability in dependence of a relaxation of the triangle inequality. In Section 31.4, we then complement the picture by introducing also a partition into infinitely many approximability classes inside the metric TSP. Moreover, lower bounds are exhibited for all these classes, inside as well as outside the metric case. They serve for judging the quality of the obtained algorithms, and they show that one cannot, in principle, expect much better results. Section 31.5 concludes the chapter with a survey of other successful applications of approximation stability and a discussion of the concept.

31.2 Definition of the Stability of Approximation Algorithms

We assume that the reader is familiar with the basic concepts and notions of algorithmics and complexity theory as presented in standard textbooks like in Refs. [13,15,23–25]. Next, we give a formal definition of the notion of an optimization problem, which is an extended version of the standard definition. This extended definition allows us to study the influence of the input sets on the hardness of the problem considered.

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of nonnegative integers, and let $\mathbb{R}^+$ be the set of positive reals.

Definition 31.1

An optimization problem *U is a 7-tuple $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$, where*

(i) Σ_I *is an alphabet called* input alphabet,
(ii) Σ_O *is an alphabet called* output alphabet,
(iii) $L \subseteq \Sigma_I^*$ *is a language over* Σ_I *called the* language of consistent inputs,
(iv) $L_I \subseteq L$ *is a language over* Σ_I *called the* language of actual inputs,
(v) $\mathcal{M}$ *is a function from* L *to* $2^{\Sigma_O^*}$, *where, for every* $x \in L$, $\mathcal{M}(x)$ *is called the* set of feasible solutions *for the input x,*
(vi) cost is a function, called cost function *that, for every pair (u, x), where $u \in \mathcal{M}(x)$ for some $x \in L$, assigns a positive real number $cost(u, x)$,*
(vii) goal $\in \{\min, \max\}$.

For every $x \in L$, we define

$$\boldsymbol{SolOpt_U(x)} = \{y \in \mathcal{M}(x) | cost(y) = goal\{cost(z) | z \in \mathcal{M}(x)\}\}$$

as the set of optimal solutions, and $\boldsymbol{Opt_U(x)} = cost(y)$ *for some* $y \in SolOpt_U(x)$.

Clearly, the meaning for Σ_I, Σ_O, $\mathcal{M}$, *cost*, and *goal* is the usual one. L may be considered as the set of consistent inputs, that is, the inputs for which the optimization problem is consistently defined. L_I is the set of inputs actually considered and only these inputs are taken into account for determining the complexity of the optimization problem U. This kind of definition is useful for considering the complexity of optimization problems parameterized according to their languages of actual inputs.

Definition 31.2

Let $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ be an optimization problem. We say that an algorithm A is a consistent algorithm for *U if, for every input $x \in L_I$, A computes an output $A(x) \in \mathcal{M}(x)$. We say that A* solves *U if, for every $x \in L_I$, A computes an output $A(x)$ from $SolOpt_U(x)$. The time complexity of A is defined as the function*

$$Time_A(n) = \max\{Time_A(x) \mid x \in L_I \cap (\Sigma_I)^n\}$$

from $\mathbb{N}$ to $\mathbb{N}$, where $Time_A(x)$ is the length of the computation of A on x.

In this chapter, we deal with the case $Time_A(n) \in O(p(n))$, for some polynomial p, only. That is, we just speak of an approximation algorithm when meaning *polynomial-time approximation algorithm.* Next, we recall the definitions of standard notions in the area of approximation algorithms.

Definition 31.3

Let $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ be an optimization problem, and let A be a consistent algorithm for U.

- *For every $x \in L_I$, the approximation ratio $R_A(x)$* of A on x *is defined as*

$$R_A(x) = \max\left\{\frac{cost(A(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(A(x))}\right\}.$$

- *For any $n \in \mathbb{N}$, we define the* approximation ratio of A *as*

$$R_A(n) = \max\{R_A(x) \mid x \in L_I \cap (\Sigma_I)^n\}.$$

For any positive real $\delta \geq 1$, we say that A is a δ-approximation algorithm for *U if $R_A(x) \leq \delta$ for every $x \in L_I$.*

- *For every function $f : \mathbb{N} \to \mathbb{R}$, we say that A is an $f(n)$*-approximation algorithm for *U if $R_A(n) \leq f(n)$ for every $n \in \mathbb{N}$.*

To define the notion of stability of approximation algorithms we need to consider something like a distance between a language L and a word outside L.

Definition 31.4

Let $U = (\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, goal)$ *and* $\overline{U} = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *be two optimization problems with* $L_I \subset L$. *A* distance function for U according to L_I *is any function* $h_L : L \to \mathbb{R}^+$ *satisfying the property*

$$h_L(x) = 0 \textit{ for every } x \in L_I$$

We define, for any $r \in \mathbb{R}^+$,

$$Ball_{r,h}(L_I) = \{w \in L \mid h(w) \leq r\}$$

Let A be a consistent algorithm for $\overline{U}$, *and let A be an* ε*-approximation algorithm for U for some* $\varepsilon \in \mathbb{R}^+$. *Let p be a positive real. We say that A is* p-stable according to h *if, for every real* $0 \leq r \leq p$, *there exists a* $\delta_{r,\varepsilon} \in \mathbb{R}^+$ *such that A is an* $\delta_{r,\varepsilon}$*-approximation algorithm for* $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.[1]

A is stable according to h *if A is p-stable according to h for every* $p \in \mathbb{R}^+$. *We say that A is* unstable according to h *if A is not p-stable for any* $p > 0$.

For every positive integer r, and every function $f_r : \mathbb{N} \to \mathbb{R}^+$ *we say that A is* $(r, f_r(n))$-quasi-stable according to h *if A is an* $f_r(n)$*-approximation algorithm for* $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.

One may see that the notion of stability can be useful for answering the question how broadly a given approximation algorithm is applicable. If one is interested in negative results, then one can try to show that for any reasonable distance measure the considered algorithm cannot be extended to work for a much larger set of inputs than the original one. In this way one can search for some more exact boundaries between polynomial approximability and polynomial inapproximability.

31.3 Stable Approximation Algorithms for the TSP

To illustrate the concept of approximation stability, we consider the well-known TSP. The TSP in its general form is very hard to approximate, but if one considers complete graphs in which the triangle inequality holds, then we have a 1.5-approximation algorithm due to Christofides [10].

To extend the part of the TSP that can be considered solvable reasonably well, it is therefore a natural idea to look at instances that "violate the triangle inequality not by much." This is formalized by the idea of the parameterized triangle inequality.

Definition 31.5

For fixed $\beta \geq \frac{1}{2}$, *a weighted graph* (G, c), $G = (V, E)$ *obeys the* β*-triangle inequality iff*

$$c(u, w) \leq \beta\,(c(u, v) + c(v, w)) \textit{ for all } u, v, w \in V$$

In case of $\beta > 1$ we speak of the *relaxed triangle inequality,* which is the the case we want to investigate in this section. The case of the *sharpened triangle inequality* is considered in Section 31.4. We call Δ_β-TSP the TSP subcase consisting of all instances obeying the β-triangle inequality (Δ_β-inequality for short).

In terms of stability, we start from Δ-TSP $= (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, \min)$, where we assume $\Sigma_I = \Sigma_O = \{0, 1, \#\}$. L contains codes of all weight functions for edges of complete graphs, and L_I contains codes of weight functions that satisfy the triangle inequality. Let, for every $x \in L$, $G_x = (V_x, E_x, c_x)$ be the complete weighted graph coded by x.

[1] Note that $\delta_{r,\varepsilon}$ is a constant depending on r and ε only.

For every TSP input outside L_I (i.e., not obeying the standard triangle inequality), we obtain a distance from L_I as follows. Let β be minimal such that the given input belongs to Δ_β-TSP, then $\beta - 1$ is the distance from L_I.

Definition 31.6

For every $x \in L$,

$$dist(x) = \max\left\{0, \max\left\{\frac{c_x(\{u, v\})}{c_x(\{u, p\}) + c_x(\{p, v\})} - 1 \,\middle|\, u, v, p \in V_x\right\}\right\}$$

Now let us look at known algorithms for Δ-TSP. We can easily observe that these algorithms work on any TSP-instance, that is, they are consistent for $(\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, \min)$. Only the guarantee for the approximation ratio depends on the triangle inequality. It is therefore natural to ask how these algorithms perform on other inputs.

We start with a 2-approximation algorithm that simply converts a doubled minimal spanning tree (MST) into a Hamiltonian path, see, for example, Ref. [16], where also a few equivalent variants are shown.

```
DOUBLE SPANNING TREE ALGORITHM
Input: A complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{R}^+$.
Step 1: $T :=$ MST of $(G, c)$.
Step 2: $\omega :=$ Eulerian tour in the multigraph obtained from $T$ by doubling each edge.
Step 3: Construct a Hamiltonian tour $H$ of $G$ by shortening $\omega$ (i.e., by removing all repetitions of the
    occurrences of every vertex in $\omega$ in one run via $\omega$ from the left to the right).
Output: $H$.
```

As each Hamiltonian tour after removing any one edge becomes a spanning tree, clearly, an MST is at most as expensive as an optimal TSP solution for the same graph. Consequently, ω costs at most twice as much as an optimal solution.

Though this holds for any TSP instance, the "shortening" in step 3 is reducing or at least not increasing the cost only under the triangle inequality. It is executed by repeatedly replacing a path $x, u_1, \ldots, u_m, y$ by the edge $\{x, y\}$ (because $u_1, \ldots, u_m$ have already occurred before in the prefix of ω). Before further pursuing that thought, let us see how the Christofides algorithm builds on the previous one.

At closer inspection it occurs that the doubling of the spanning tree was needed only to get an Eulerian subgraph. Consequently, in the Christofides algorithm the idea is to add only one edge each to the vertices of odd degree in T to get an Eulerian subgraph as cheap as possible.

```
CHRISTOFIDES ALGORITHM
Input: A complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{R}^+$.
Step 1: $T :=$ MST of $(G, c)$.
Step 2: $S := \{v \in V \mid deg_T(v)$ is odd$\}$.
Step 3: Compute a minimum-weight perfect matching $M$ on $S$ in $G$.
Step 4: $\omega :=$ Eulerian tour in the multigraph $G' = (V, E(T) \cup M)$.
Step 5: Construct a Hamiltonian tour $H$ of $G$ by shortening $\omega$.
Output: $H$.
```

In addition to the above reasoning for the Double Spanning Tree Algorithm, here we have to convince ourselves that M costs at most half as much as an optimal TSP solution. The reasoning is that by identifying the vertices of S in an optimal solution H_{opt}, we cut the circle H_{opt} at every vertex from S. Assigning the resulting paths segments alternately into two sets and shortening each into single edges, we obtain two matchings M_1, M_2. Obviously, M_1, M_2 each cost as least as much as M.

But from this, $cost(M) \leq \frac{1}{2} cost(H_{opt})$ follows only under triangle inequality since we had to shorten the path segments from H_{opt} into single edges to apply our argument.

We can see how the Christofides algorithm depends on the triangle inequality even more intricately than the Double Spanning Tree Algorithm. Here, the inequality is not only used in a constructive step but also already in the argument to show that the initial component M of the solution is suitable.

When applying these two algorithms to input instances from Δ_β-TSP for $\beta > 1$, one has to expect the approximation ratio to grow at least with β since β factors immediately in the "shortening" of paths. Unfortunately, however, for $\beta > 1$, the approximation ratio becomes dependent on the input size as we will see now.

Lemma 31.1

The Christofides Algorithm is unstable for dist.

More precisely, for every $r \in \mathbb{R}^+$, if the Christofides algorithm is $(r, f_r(n))$-quasi-stable for dist., then

$$f_r(n) \geq n^{\log_2(1+r)}/(2 \cdot (1+r))$$

Remember that the distance r from Δ-TSP of an input x is the minimal $\beta - 1$ such that x is a Δ_β-TSP instance, that is, we may use $\beta = 1 + r$ in the following as a shorthand.

Proof

We construct a weighted complete graph from $Ball_{r,dist}(L_I)$ as follows (Figure 31.1). We start with the path $p_0, p_1, \ldots, p_n$ for $n = 2^k$, $k \in \mathbb{N}$, where every edge $\{p_i, p_{i+1}\}$ has weight 1. Then we add edges $\{p_i, p_{i+2}\}$ for $i = 0, 1, \ldots, n-2$ with weight $2 \cdot \beta$. Generally, for every $m \in \{1, \ldots, \log_2 n\}$, we define $c(\{p_i, p_{i+2^m}\}) = 2^m \cdot \beta^m$ for $i = 0, \ldots, n - 2^m$. For all other edges one can take maximal possible weights in such a way that the constructed input is in $Ball_{r,dist}(L_I)$.

Let us have a look on the work of the Christofides Algorithm on the input (G, c). There is only one MST that corresponds to the path containing all edges of weight 1 (Figure 31.1). Since every path contains exactly two vertices of odd degree, the Eulerian graph constructed in step 4 is the cycle $D = p_0, p_1, p_2, \ldots, p_n, p_0$ with the n edges of weight 1 and the edge of the maximal weight $n \cdot \beta^{\log_2 n} = n^{1+\log_2 \beta}$. Since the Eulerian tour is a Hamiltonian tour (Figure 31.1), the output of the Christofides algorithm is unambiguously the cycle $p_0, p_1, \ldots, p_n, p_0$ with cost $n + n\beta^{\log_2 n}$. The optimal tour for this input is

$$H_{Opt} = p_0, p_2, p_4, \ldots, p_{2i}, p_{2(i+1)}, \ldots, p_n, p_{n-1}, p_{n-3}, \ldots, p_{2i+1}, p_{2i-1}, \ldots, p_3, p_1, p_0$$

This tour contains two edges $\{p_0, p_1\}$ and $\{p_{n-1}, p_n\}$ of weight 1 and all $n - 2$ edges of weight $2 \cdot \beta$. Thus, $cost(H_{Opt}) = 2 + 2 \cdot \beta \cdot (n-2)$ and

$$\frac{cost(D)}{cost(H_{Opt})} = \frac{n + n \cdot \beta^{\log_2 n}}{2 + 2 \cdot \beta \cdot (n-2)} \geq \frac{n^{1+\log_2 \beta}}{2n \cdot \beta} = \frac{n^{\log_2 \beta}}{2\beta} \qquad \square$$

Roughly the same happens for the Double Spanning Tree Algorithm. Only the fact that the choice of the root of the MST is not fixed may allow that algorithm to gain a factor of β at best (if two edges replacing paths of about $n/2$ edges each are used).

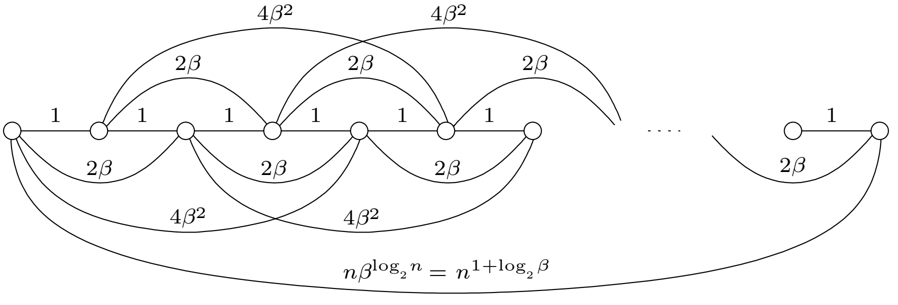


FIGURE 31.1 Unstability of the Christofides algorithm.

One may observe that in this example, the Double Spanning Tree Algorithm can fare much better if, in shortening the Eulerian tour into a Hamiltonian one, a more clever strategy is used. The Eulerian tour is in this case just a double traversal of the path that forms the MST. If on each traversal every other vertex is used, an optimal solution will be obtained. This idea might not work on every MST, but it is the basis for the first Δ_β-TSP algorithm by Andreae and Bandelt discussed below.

Afore, we just note that the above example marks roughly the worst case for both algorithms.

Lemma 31.2

For every positive real number r, the Christofides Algorithm and the Double Spanning Tree Algorithm are $(r, O(n^{\log_2((1+r)^2)}))$-quasi-stable for dist.

Now we turn to the question how to modify those algorithms to get algorithms that are stable according to *dist*.

As we have observed, the main problem is that shortening a path $u_1, u_2, \ldots, u_{m+1}$ to the edge u_1, u_{m+1} can lead to

$$cost(\{u_1, u_{m+1}\}) = \beta^{\lceil \log_2 m \rceil} \cdot cost(u_1, u_2, \ldots, u_{m+1})$$

Obviously, we need to limit by a constant the number of consecutive edges replaced by a single new edge. For a spanning tree being just a path, we have mentioned that it suffices to replace at most two consecutive edges. For a general spanning tree, the following result implies that replacing at most three consecutive edges suffices.

We use, for any graph $G = (V, E)$ and $k \in \mathbb{N}_{\geq 2}$, the notation

$$G^k = (V, \{\{x, y\} \mid x, y \in V, \text{ there is a path } x, P, y \text{ in } T \text{ of a length at most } k\})$$

Theorem 31.1 (Sekanina [26])

For every tree $T = (V, E)$, the graph T^3 contains a Hamiltonian tour H.

This means that every edge $\{u, v\}$ of H has a corresponding unique path $u, P_{u,v}, v$ in T of length at most three. This is a positive development, but it still does not solve our problem completely. The remaining task is that we need to estimate a good upper bound on the cost of the path

$$P(H) = u_1, P_{u_1,u_2}, u_2, P_{u_2,u_3}, u_3, \ldots, u_{n-1} P_{u_{n-1},u_n}, u_n, P_{u_n,u_1}, u_1$$

(in T) that corresponds to the Hamiltonian tour $u_1, u_2, \ldots, u_n, u_1$ in T^3.

The problem is that we do not know the frequency of the occurrences of particular edges of T in $P(H)$. It may happen that the most expensive edges of T occur more frequently in $P(H)$ than the cheap edges. Observe also that $cost(T^3)$ cannot be bounded by $c \cdot cost(T)$ for any constant c independent on T, because T^3 may be even a complete graph for some trees T. Thus, we need the following refinement of the above theorem which is obtained by a slight modification of the original proof.

Lemma 31.3 (Andreae and Bandelt [27])

Let T be a tree with $n \geq 3$ vertices, and let $\{p, q\}$ be an edge of T. Then, T^3 contains a Hamiltonian path $U = v_1, v_2, \ldots, v_n$, $p = v_1$, $v_n = q$, such that every edge of $E(T)$ occurs exactly twice in $P_T(H)$, where $H = U, p$ is a Hamiltonian tour in T^3.

Let us call a Hamiltonian path of the type described in the lemma *admissible*. Then we have the following refinement of the Double Spanning Tree Algorithm, developed by Andreae and Bandelt [27].

```
T^3-ALGORITHM
Input: A complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{R}^+$.
Step 1: $T :=$ MST of $(G, c)$.
Step 2: Construct an admissible Hamiltonian tour $H$ of $T$.
Output: $H$.
```

Originally, the approximation ratio was estimated to be $\frac{3}{2}\beta^2 + \frac{1}{2}\beta$, but by further refining step 2, Andreae [28] could lower this a bit.

Theorem 31.2 **(Andreae and Bandelt, and Andreae [27,28])**

The refined T^3-algorithm is a $\beta^2 + \beta$-approximation algorithm for Δ_β-TSP, and it runs in time $O(n^2)$ on a graph with n vertices.

Taking into account the fact that T^2 does not always contain a Hamiltonian tour, if one wants to get below the β^2 factor one needs to consider a suitable replacement for a spanning tree as starting point.

This is the approach followed by Bender and Chekuri [29], building on a 2-vertex-connected spanning subgraph (2VCSS) of G.

Theorem 31.3 **(Fleischner [30])**

For every 2-vertex-connected graph S, the graph S^2 contains a Hamiltonian tour H.

As desired, now we need to replace only two consecutive edges by one. However, as opposed to an MST, a minimal spanning 2VCSS of G cannot be computed in polynomial time, unless P=NP. Therefore, one has to use an approximation instead.

Theorem 31.4 **(Penn and Shasha-Krupnik [31])**

There is a polynomial-time 2-approximation algorithm for the minimal 2VCSS problem.

Additionally, as for the T^3-algorithm, the existence of a Hamiltonian tour is not enough. From a constructive version of Fleischner's proof by Lau [32], one can get an 8β-approximation. By refining this such that the resulting Hamiltonian tour uses every edge of S at most twice, one gets the following result. Again, we call such a tour *admissible.*

```
$S^2$-ALGORITHM
Input: A complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{R}^+$.
Step 1: $S :=$ 2-approximation of a minimal 2VCSS of $(G, c)$.
Step 2: Construct an admissible Hamiltonian tour $H$ of $S$.
Output: $H$.
```

Theorem 31.5 **(Bender and Chekuri [29])**

The S^2-Algorithm is a 4β-approximation algorithm for Δ_β-TSP, and it runs in time $O(n^5)$ on a graph with n vertices.

The impractical running time is clearly a drawback. It comes from computing the 2VCSS. Obviously, any improvement on the 2VCSS approximation would also improve the S^2-algorithm, w.r.t. running time or approximation ratio. However, since the algorithm by Penn and Shasha-Krupnik is already the result on several improvements in smaller and smaller steps, one cannot be too optimistic in that regard. At the current state, the S^2-algorithm surpasses the much faster T^3-algorithm only for $\beta > 3$, at which point the approximation ratio already is 12.

Another downside to both algorithms is that for inputs close to Δ-TSP they cannot get close to the results by the Christofides algorithm. Especially, for instances where only a few edge costs violate the triangle inequality, an algorithm based on the Christofides algorithm should in most cases get close to or reach the 1.5-approximation ratio, where S^2- and T^3-algorithms can only hope to return about a 2-approximation in such cases.

The difficulty, as mentioned before, is that the Christofides algorithm uses a matching as one of its starting points, and the estimate that this does not cost more than half of a Hamiltonian tour fails if the triangle inequality does not hold. Therefore, it was even conjectured in Ref. [27] that the Christofides algorithm cannot be adopted to Δ_β-TSP for $\beta > 1$.

However, there is a way around this by computing a so-called path matching.

Definition 31.7

For a weighted graph (G, c), $G = (V, E)$ and an even set $W \subseteq V$, a path matching *is a set M of paths in G such that each vertex from W is the endpoint of exactly one path from M.*

A path matching of minimal cost can be computed in time $|V|^3$. First, one solves the all-pairs cheapest path problem on (G, c) and let $c'(u, v)$ be the cost of cheapest path between u and v. Then one computes a minimum matching M' for (G, c'), and M is obtained from M' by replacing every edge u, v by the cheapest path from u to v computed previously on (G, c).

It is easy to see that for every TSP instance, independent of the triangle inequality, a minimum path matching costs at most half as much as an optimum TSP solution. This holds because an optimum TSP solution can simply be divided into two path matchings.

Now that this initial problem is solved, one is left with a combination of spanning tree T and path matching M. This is a structure where no result like the ones of Sekanina and Fleischner is known. To construct in the end a Hamiltonian path without replacing more than four consecutive edges by a new one, one needs some detail work additionally. ("PMCA" stands for path matching and the Christofides algorithm-based approach.)

```
ALGORITHM PMCA [33]
Input: A complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{R}^+$.
Step 1: $T := $ MST of $(G, c)$.
Step 2: $S := \{v \in V \mid deg_T(v)$ is odd$\}$.
Step 3: Compute a minimum-weight path matching $M_1$ on $S$ in $G$.
Step 4: Replace $M_1$ by vertex-disjoint path matching $M_2$.
Step 5: $\omega_1 := $ Eulerian tour in the multigraph $G' = (V, E(T) \cup M_2)$.
Step 6: Modify parts from $T$ in $\omega_1$ such that they form a forest of degree at most 3, obtain $\omega_2$
Step 7: Construct a Hamiltonian tour $H$ of $G$ by shortening $\omega_2$.
Output: $H$.
```

The detail work happens in steps 4, 6, and 7.

While any minimum-weight path matching can easily be shown to be edge-disjoint, for obtaining vertex-disjointness in step 4 a price has to be paid. At vertices used by two paths, one of them has to give way by "bridging" the two edges to and from that vertex, that is, replacing them by a new one. This can be done such that a new edge is never replaced again later. Thus, the costs increase at most by β, that is, $cost(M_2) \leq \beta cost(M_1)$.

Similarly, we look in step 6 at partial paths from ω_1 in T, that is, at those pieces we obtain after dividing ω_1 into parts from T and parts from M_2 alternately. At a vertex of degree ≥ 4 in T, two or more path pieces from ω_1 in T will cross, and we can bridge some of them like in step 4, replacing at most two consecutive edges from T by a new one. Thus, we have

$$cost(\omega_2) \leq cost(M_2) + \beta cost(T) \leq \beta\,(cost(M_1) + cost(T))$$

After this preparation, in ω_2 each vertex will occur not more than twice, and bridging one of the occurrences for each such vertex is performed in step 7 again by replacing at most two consecutive edges from ω_2 by a new one. This gives the resulting estimate, where H_{opt} is an optimal TSP solution.

$$cost(H) \leq \beta cost(\omega_2) \leq \beta^2\,(cost(M_1) + cost(T)) \leq \beta^2 \cdot 1.5 cost(H_{opt})$$

Theorem 31.6 **(Böckenhauer et al. [33])**

The Algorithm PMCA is a $1.5\beta^2$-approximation algorithm for Δ_β-TSP, and it runs in time $O(n^3)$ on a graph with n vertices.

This algorithm has a practical running time, and it gives the best approximation ratio so far for cases $\beta \leq 2$. Note that for $\beta = 2$ the approximation ratio of algorithms PMCA and T^3 is 6 already. Thus algorithm PMCA covers the cases close to the kernel L_I that are in practice the most interesting ones.

Probably, it is the most prominent open question in this area whether this algorithm can be improved to ratio 1.5β, respectively whether another approach can reach that goal. Certainly, one should not hope for anything better in view of the lower bounds discussed in the next section.

Let us finish this section by summarizing what we have seen by studying stability for the TSP, this archetype of a hard problem. It has been very fruitful, starting from known algorithms for a kernel problem, first to identify how they depend on properties of that kernel, and then to look for modifications or similar approaches. There, the crucial point was to transform a dependency of a kernel property into one that holds for each distance from the kernel, albeit with a worse ratio but independent of the input size. One can see that different tools may come into play, modifying the starting point, using general graph properties, or fine-tuning the algorithm's behavior at points where it did not matter for the original kernel problem.

31.4 Lower Bounds and the Situation Inside Δ-TSP

We have seen in the previous section how to use the concept of stability to partition all TSP instances into infinitely many classes according to their approximability, depending on the parameter β of a relaxed β-triangle inequality. Our aim in this section is twofold: We will first exhibit a similar partition into infinitely many classes also for the input instances inside the Δ-TSP, based on the parameter β, and we will show that the concept of stability can also be used to obtain lower bounds on the approximability for all of these classes.

Recall that for a complete edge-weighted graph $G = (V, E, c)$, we say that it satisfies a *sharpened triangle inequality*, if, for all $u, v, w \in V$,

$$c(\{u, v\}) \leq \beta \cdot (c(\{u, w\}) + c(\{w, v\})$$

holds for some $\frac{1}{2} \leq \beta < 1$. Note that $\beta = \frac{1}{2}$ corresponds to the trivial case where all edge weights are equal.

In Ref. [34] it was shown how to estimate the approximation ratio of any approximation algorithm for Δ-TSP on instances of Δ_β-TSP for $\frac{1}{2} < \beta < 1$.

Theorem 31.7 **(Böckenhauer et al. [34])**

Let A be an approximation algorithm for Δ-TSP with approximation ratio α, and let $\frac{1}{2} < \beta < 1$.

Then A can be used as an approximation algorithm for Δ_β-TSP with approximation ratio $\frac{\alpha \cdot \beta^2}{\beta^2 + (\alpha - 1) \cdot (1 - \beta)^2}$.

The proof of Theorem 31.7 is based on the following idea: For any input instance for the Δ_β-TSP, we can subtract a certain amount (depending on β and the value of the minimum edge weight) from all edge weights such that the resulting TSP instance still satisfies the triangle inequality. Furthermore, the optimal Hamiltonian tours for both instances coincide. Using the observation that, in a weighted graph obeying a sharpened β-triangle inequality, the maximum edge weight is bounded from above by $\frac{2\beta^2}{1-\beta}$ times the minimum edge weight, we can then estimate the length of a Hamiltonian tour as computed by algorithm A as stated in Theorem 31.7.

Note that the approximation ratio guaranteed by Theorem 31.7 tends to 1 for β tending to $\frac{1}{2}$, and it tends to α for β tending to 1. Applied to the Christofides algorithm, we obtain an approximation ratio of $1 + \frac{2\beta - 1}{3\beta^2 - 2\beta + 1}$.

For $\frac{1}{2} \leq \beta < \frac{2}{3}$, this result can be improved by a specific algorithm, called Cycle Cover Algorithm, which we will explain below.

Theorem 31.8 **(Böckenhauer et al. [34])**

For $\frac{1}{2} \leq \beta < 1$, the Cycle Cover Algorithm is a $(\frac{2}{3} + \frac{1}{3} \cdot \frac{\beta}{1-\beta})$-approximation algorithm for Δ_β-TSP.

This algorithm is based on the idea of first computing a minimum cycle cover of the given graph, this can be done in polynomial time [35]. Then the cycles are joined together to form a Hamiltonian tour by removing the cheapest edge from every cycle and replacing it by an edge to the next cycle. One can easily observe that, in a graph satisfying a sharpened β-triangle inequality, for any two adjacent edges the costs differ by a factor of at most $\frac{\beta}{1-\beta}$. Since every cycle of the cover has at least three edges, this possible growth of the costs affects at most one third of the total cost of the cycle cover, which gives the claimed approximation ratio.

We have seen so far that we can partition the set of TSP input instances into infinitely many classes according to their approximability. We have used the relaxed and sharpened triangle inequality to define these classes. In the following we will show that it is also possible to give an explicit lower bound on the approximability of the TSP for each of these classes, for any $\beta > \frac{1}{2}$. These lower bounds imply that, unless P $=$ NP, one can expect to obtain only gradually better results, not principally better ones. Specifically, as we will see, no polynomial-time approximation scheme can exist for Δ_β-TSP for $\beta > \frac{1}{2}$, even when getting arbitrarily close to the trivial case of $\beta = \frac{1}{2}$. Also, for growing β, we will see the necessity for the approximation ratio of any polynomial-time algorithm to grow at least linearly with β.

The first in-approximability proof for the metric TSP, that is, for $\beta = 1$, goes back to Papadimitriou and Yannakakis [36], who proved that even TSP restricted to edge weights from the set {1, 2} does not admit a PTAS.

The first explicit lower bound on the approximability of the metric TSP was given by Engebretsen [37], who proved that it is NP-hard to approximate the Δ-TSP within a factor of $\frac{5381}{5380} - \varepsilon$ for any small $\varepsilon > 0$, even in the case where all edge weights are from {1, 2}.

This result was not only improved for the metric case, but also extended to the cases of a relaxed or sharpened triangle inequality by Böckenhauer and Seibert [38], who proved the following theorem.

Theorem 31.9 **(Böckenhauer and Seibert [38])**

Unless P≠NP, there is no polynomial-time α-approximation algorithm, if

$$\alpha < \begin{cases} \dfrac{7611 + 10\beta^2 + 5\beta}{7612 + 8\beta^2 + 4\beta} & \textit{for the case } \dfrac{1}{2} < \beta \leq 1 \\ \dfrac{3803 + 10\beta}{3804 + 8\beta} & \textit{for the case } \beta \geq 1 \end{cases}$$

For the metric TSP, that is, for $\beta = 1$, this leads to a lower bound of $\frac{3813}{3812}$. The currently best known lower bound of $\frac{220}{219}$ for the metric TSP is due to Papadimitriou and Vempala [39]. However, since in Ref. [39] edge weight 0 was used, this result cannot be adapted to Δ_β-TSP for $\beta < 1$.

The proof of Theorem 31.9 is based on a gap-preserving reduction from the LinEq2-2(3) problem.[2] LinEq2-2(3) is the following problem: Given a system of linear equations modulo 2 with exactly two variables in each equation, and with exactly three occurrences of each variable, find the maximum number of equations that can be simultaneously satisfied. This problem was proven to be not approximable in polynomial time within $\frac{332}{331} - \varepsilon$ for an arbitrarily small $\varepsilon > 0$ by Berman and Karpinski [41].

The idea of the reduction is based on building so-called gadgets. For each equation and for each variable a small graph is built, and all these graphs are put together into a large graph that serves as the input for the TSP. In this construction, only three different edge weights are used, such that, for each nonsatisfied equation in an optimal variable assignment, the optimal Hamiltonian tour through the constructed graph has to pass through one edge of highest cost. All other edges of an optimal tour are shown to be of lowest cost.

Theorem 31.9 is most interesting for the case $\beta < 1$. It shows the strong result that the TSP with sharpened triangle inequality does not admit a PTAS even if β comes arbitrarily close to the trivial case $\frac{1}{2}$.

For the case of a relaxed triangle inequality, the result from Theorem 31.9 tends to $\frac{5}{4}$ for β tending to infinity. Bender and Chekuri [29] extended the result from Ref. [36] to the case of the relaxed triangle

[2] For an introduction into the theory of gap-preserving reductions see Chapters 15 and 17 or, for a broader treatment, see Ref. [40].

inequality and showed a lower bound of $1 + \varepsilon\beta$ for some very small ε, not given explicitly in the proof. This result shows that the approximation ratio of any polynomial-time algorithm for the Δ_β-TSP has to grow at least linearly with β.

31.5 Discussion and Related Work

In the previous sections we have introduced the concept of stability of approximation. Here we discuss the potential applicability and usefulness of this concept.

Using this concept, one can establish positive results of the following types:

(1) An approximation algorithm or a PTAS can be successfully used for a larger set of inputs than the set usually considered.
(2) We are not able to successfully apply a given approximation algorithm A (a PTAS) for additional inputs, but we can simply modify A to get a new approximation algorithm (a new PTAS) working for a larger set of inputs than the original set of inputs of A.
(3) To learn that an approximation algorithm is unstable for a distance measure could lead to the development of completely new approximation algorithms that would be stable according to the considered distance measure.

The following types of negative results may be achieved:

(1) The fact that an approximation algorithm is unstable according to all "reasonable" distance measures and so that its use is really restricted to the original input set.
(2) Let $Q = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal) \in NPO$ be well approximable. If, for a distance measure D and a constant r, one proves the nonexistence of any polynomial-time approximation algorithm for $Q_{r,D} = (\Sigma_I, \Sigma_O, L, Ball_{r,D}(L_I), \mathcal{M}, cost, goal)$, then this means that the problem Q is "unstable" according to D.

Thus, using the notion of stability one can search for a spectrum of the hardness of a problem according to the set of inputs. For instance, considering a hard problem such as the TSP or the Clique problem, one could get an infinite sequence of input languages $L_0, L_1, L_2, \ldots$ given by some distance measure, where $R_r(n)$ is the best achievable approximation ratio for the language L_r. Results of this kind can essentially contribute to the study of the nature of hardness of specific problems.

In case of the TSP with relaxed triangle inequality, this approach has been followed successfully as seen in the preceding sections. A variety of new and adapted algorithms has been developed after the known ones have proven to be unstable. All in all, an approximation ratio of $\min\{\beta^2 + \beta, \frac{3}{2}\beta^2, 4\beta\}$ has been reached so far, and the proven lower bounds show the β-dependency to be invariable.

Further applications of these ideas for different versions of the Hamiltonian path problem were developed by Forlizzi et al. in Ref. [42], where a few stable algorithms with respect to relaxed triangle inequality were designed.

Also we have seen how these studies gave rise to looking into subcases inside the metric one. Though not a direct application of the stability concept, the treatment here was inspired by the stability approach, and it has proven successful, too. The combined approximation ratio in this case was $\min\{\frac{2}{3} + \frac{1}{3} \cdot \frac{\beta}{1-\beta}, 1 + \frac{2\beta-1}{3\beta^2-2\beta+1}\}$.

Moreover, Chandran and Ram [43], and then Bläser [44], succeeded in transferring this approach to the asymmetric TSP. Remember that no constant-factor approximation algorithm for the metric case is known, so the combined approximation factor of $\min\{\frac{\beta}{1-\beta}, \frac{1}{1-\frac{1}{2}(\beta+\beta^2)}\}$ is quite an achievement in the asymmetrical case.

The subproblems with sharpened triangle inequality were also successfully attacked for a variety of minimum connected spanning subgraph problems in Refs. [45,46]. Since these problems allow constant approximation algorithms in their general case, a treatment analogous to the TSP could not be expected.

However, the approach guided by the parameterized triangle inequality led to the development of a few new simple and fast algorithms that improve the best known approximation ratio at least for a part of the sharpened triangle inequality case.

All in all, we have seen that the stability of approximation approach leads to rethinking known algorithms and developing new ones. The common goal is to provide a variety of algorithms for the whole range of a problem that deliver approximation ratios depending not on the input size but on a suitable parameter indicating the hardness of each input instance.

References

[1] Graham, R. L., Bounds for certain multiprocessing anomalies, *Bell Syst. Tech. J.,* 45, 1563, 1966.

[2] Graham, R. L., Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.,* 17, 263, 1969.

[3] Cook, S. A., The complexity of theorem proving procedures, *Proc. of STOC*, 1971, p. 151.

[4] Karp, R. M., Reducibility among combinatorial problems, in *Complexity of Computer Computations*, Miller, R. E. and Thatcher, J. W., Eds., Plenum Press, New York, 1972, p. 85.

[5] Johnson, D. S., Approximation algorithms for combinatorial problems, *JCSS*, 9, 256, 1974.

[6] Sahni, S. and Gonzalez, T. F., P-complete approximation problems, *JACM*, 23, 555, 1976.

[7] Rosenkrantz, R., Stearns, R., and Lewis, L., An analysis of several heuristics for the traveling salesman problem, *SIAM J. Comput.,* 6(3), 563, 1977.

[8] Lovász, L., On the ratio of the optimal integral and functional covers, *Disc. Math.*, 13, 383, 1975.

[9] Ibarra, O. H. and Kim, C. E., Fast approximation algorithms for the knapsack and sum of subsets problem, *JACM*, 22, 463, 1975.

[10] Christofides, N., Worst-case analysis of a new heuristic for the travelling salesman problem, Technical report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976.

[11] Hromkovič, J., *Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*, Springer, Berlin, 2003.

[12] Vazirani, V. V., *Approximation Algorithms*, Springer, Berlin, 2003.

[13] Hochbaum, D. S., Ed., *Approximation Algorithms for NP-hard Problems*, PWS Publishing Company, Boston, 1996.

[14] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M., *Complexity and Approximation—Combinatorial Optimization Problems and Their Approximability Properties*, Springer, Berlin, 1999.

[15] Garey, M. R. and Johnson, D. S., *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.

[16] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B., Eds., *The Traveling Salesman Problem*, Wiley, New York, 1985.

[17] Papadimitriou, Ch., The Euclidean travelling salesman problem is NP-complete, *Theor. Comp. Sci.*, 4, 237, 1977.

[18] Arora, S., Polynomial time approximation schemes for Euclidean TSP and other geometric problems, *Proc. of FOCS*, 1996, p. 2.

[19] Arora, S., Nearly linear time approximation schemes for Euclidean TSP and other geometric problems, *Proc. of FOCS*, 1997, p. 554.

[20] Mitchell, J. S. B., Guillotine subdivisions approximate polygonal subdivisions: Part II—a simple polynomial-time approximation scheme for geometric k-MST, TSP and related problems, Technical report, Department of Applied Mathematics and Statistics, Stony Brook, 1996.

[21] Downey, R. G. and Fellows, M. R., Fixed-parameter tractability and completeness I: basic Results, *SIAM J. Comput.*, 24, 873, 1995.

[22] Downey, R. G. and Fellows, M. R., *Parametrized Complexity*, Springer, Berlin, 1999.

[23] Bovet, D. P. and Crescenzi, C., *Introduction to the Theory of Complexity*, Prentice-Hall, New York, 1993.

[24] Papadimitriou, Ch., *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[25] Hromkovič, J., *Theoretical Computer Science*, Springer, Berlin, 2004.

[26] Sekanina, M., On an ordering of a set of vertices of a connected graph, *Publ. Fak. Sci. Univ. Brno*, 412, 137, 1960.

[27] Andreae, T. and Bandelt, H.-J., Performance guarantees for approximation algorithms depending on parameterized triangle inequalities, *SIAM J. Disc. Math.*, 8, 1, 1995.

[28] Andreae, T., On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality, *Networks*, 38, 59, 2001.

[29] Bender, M. and Chekuri, C., Performance guarantees for TSP with a parametrized triangle inequality, *Inf. Proc. Lett.*, 73, 17, 2000.

[30] Fleischner, H., The square of every two-connected graph is Hamiltonian, *J. Comb. Theor. B*, 16, 29, 1974.

[31] Penn, M. and Shasha-Krupnik, H., Improved approximation algorithms for weighted 2- and 3-vertex connectivity augmentation problems, *J. Algorithms*, 22, 187, 1997.

[32] Lau, H. T., Finding EPS-graphs, *Monatshefte Mathematik*, 92, 37, 1981.

[33] Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., and Unger, W., Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem, *Theor. Comp. Sci.*, 285, 3, 2002.

[34] Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., and Unger, W., Approximation algorithms for TSP with sharped triangle inequality, *Inf. Proc. Lett.*, 75, 133, 2000.

[35] Edmonds, J. and Johnson, E. L., Matching: A well-solved class of integer linear programs, *Proc. Calgary Int. Conf. on Comb. Struct. and their Appl.*, Gordon and Breach, New York, 1970, p. 89.

[36] Papadimitriou, Ch. and Yannakakis, M., The traveling salesman problem with distances one and two, *Math. Oper. Res.*, 18, 1, 1993.

[37] Engebretsen, L., An explicit lower bound for TSP with distances one and two. in *Proc. of STACS*, Lecture Notes in Computer Science, Vol. 1563, Springer, Berlin, 1999, p. 373. Full version in: *Electronic Colloquium on Computational Complexity*, (http://www.eccc.uni-trier.de/eccc/), Rev. 1 of Rep. 46, 1999.

[38] Böckenhauer, H.-J. and Seibert, B., Improved lower bounds on the approximability of the traveling salesman problem, *RAIRO Theor. Inform. Appl.*, 34, 213, 2000.

[39] Papadimitriou, Ch. and Vempala, S., On the approximability of the traveling salesman problem, *Proc. STOC*, 2000. Corrected full version available at http://www.cs.berkeley.edu/˜christos/

[40] Mayr, E. W., Prömel, H. J., and Steger, A., Eds., *Lecture on Proof Verification and Approximation Algorithms*, Lecture Notes in Computer Science, Vol. 1967, Springer, Berlin, 1998.

[41] Berman, P. and Karpinski, M., On some tighter inapproximability results, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 1644, Springer, Berlin, 1999, p. 200.

[42] Forlizzi, L., Hromkovič, J., Proietti, G., and Seibert, S., On the stability of approximation for Hamiltonian path problems, *Proc. Theory and Practice of Comp. Sci.*, Lecture Notes in Computer Science, Vol. 3381, Springer, Berlin, 2005, p. 147.

[43] Chandran, L. S. and Ram, L. S., Approximations for ATSP with parametrized triangle inequality, *Proc. STACS*, Lecture Notes in Computer Science, Vol. 2285, Springer, Berlin, 2002, p. 227.

[44] Bläser, M., An improved approximation algorithm for the asymmetric TSP with strengthened triangle inequality, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 2719, Springer, Berlin, 2003, p. 157.

[45] Böckenhauer, H.-J., Bongartz, D., Hromkovič, J., Klasing, R., Proietti, G., Seibert, S., and Unger, W., On the hardness of constructing minimal 2-connected spanning subgraphs in complete graphs with sharped triangle inequality, *Theor. Comput. Sci.*, 326, 137, 2004.

[46] Böckenhauer, H.-J., Bongartz, D., Hromkovič, J., Klasing, R., Proietti, G., Seibert, S., and Unger, W., On *k*-connectivity problems with sharpened triangle inequality, *Proc. Italian Conf. on Algorithms and Complexity*, Lecture Notes in Computer Science, Vol. 2653, Springer, Berlin, 2003, p. 189.

IV

Traditional Applications

32

Performance Guarantees for One-Dimensional Bin Packing

Edward G. Coffman, Jr.
Columbia University

János Csirik
University of Szeged

32.1 Introduction

Let $a_1, \ldots, a_n$ be a given collection of *items* with *sizes* $s(a_i) > 0,\ 1 \leq i \leq n$. In mathematical terms, *bin packing* is a problem of partitioning the set $\{a_i\}$ under a sum constraint: Divide $\{a_i\}$ into a minimum number of blocks, called *bins*, such that the sum of the sizes of the items in each bin is at most a given *capacity* $C > 0$. To avoid trivialities, it is assumed that all item sizes fall in $(0, C]$. Research into bin packing and its many variants, which began some 35 years ago [1,2], continues to be driven by a countless variety of applications in the engineering and information sciences. The following examples give an idea of the scope of the applications:

- (Stock cutting) Lumber with a fixed cross section comes in a standard board C units in length. The items are demands for pieces that must be cut from such boards. The objective is to minimize the number of boards (bins) used for the pieces $\{a_i\}$, or equivalently, to minimize the *trim loss* or waste (the total board length used minus the sum of the $s(a_i)$). It is easy to see that this type of application extends to industries that supply cable, tubing, cord, tape, and so on from standard lengths.
- (Television programming) Fixed duration time slots are provided between segments of entertainment programs for the use of commercials. The objective is to minimize the number of time slots (bins) that need to be devoted to a given collection of variable length commercials $\{a_i\}$.
- (Transportation) The a_i are items to be loaded onto a collection of identical transports (e.g., trucks, railway cars, and airplanes) with given weight and volume limits. The objective is to minimize the number of transports (bins) needed; if only the weight limit is operative then the $s(a_i)$ denote weights, and if only the volume limit is operative then the $s(a_i)$ represent volumes.
- (Computer storage allocation) In this application, the items are files to be stored on a set of identical disks with the constraint that each file must be stored entirely on one disk; and the objective is to minimize the number of disks (bins) needed for the set of files.

In the above applications, bin packing has been presented as the *primary* combinatorial problem. In many applications, it is a secondary problem or an embedded special case. For example, capacitated vehicle routing is a classic problem in which bin packing is embedded.

Bin packing is an NP-hard problem, so a large majority of the research on bin packing focuses on the design and analysis of approximation algorithms. Apart from applications, bin packing has acquired an added, more general importance in the development of complexity theory and algorithms. It was one of a relatively few "test" problems, such as satisfiability and chromatic index, in which reducibility arguments were most often formulated in NP-completeness proofs. Bin packing has also served as a testbed for advances in algorithmics such as approximation schemes and average-case analysis.

We normalize the problem by dividing all item sizes by C and letting the bin capacity be 1. Let $A(L)$ denote the number of bins needed by algorithm A to pack the items of L. The symbol *OPT* stands for an optimal algorithm. Define V_α as the set of all lists of items with sizes no larger than α and consider, for given k and α, the upper bound

$$R_A(k, \alpha) := \sup\left\{ \frac{A(L)}{k} : L \in V_\alpha \text{ and } \mathrm{OPT}(L) = k \right\}$$

The limit $k \to \infty$ of this bound is the *asymptotic worst-case ratio* for algorithm A:

$$R_A^\infty(\alpha) := \limsup_{k \to \infty} R_A(k, \alpha)$$

A less formal, but more instructive, definition states that $R_A^\infty(\alpha)$ is the smallest constant such that there exists a constant $K < \infty$ for which

$$A(L) \le R_A^\infty(\alpha) \cdot \mathrm{OPT}(L) + K$$

for every list $L \in V_\alpha$; the asymptotic ratio, a multiplicative constant, hides the additive constant K. This ratio is of most interest in those applications where K is small relative to $A(L)$.

The effect of the additive constant is preserved in the *absolute worst-case ratio* for algorithm A:

$$R_A(\alpha) = \sup_{L \in V_\alpha} \left\{ \frac{A(L)}{\mathrm{OPT}(L)} \right\} \tag{32.1}$$

An algorithm A is either *online* or *offline*. If online, A assigns items to bins in the order they are given in the original list, without using any knowledge about subsequent items in the list. If offline, the entire list is available to A as it computes the packing.

Online algorithms may be the only ones that can be used in certain situations, where the items arrive in sequence according to some physical process and must be assigned to a bin at arrival time. In many cases an algorithm is offline only in that it performs an initial ordering of the items before applying an online rule. An algorithm is *bounded-space* if the bins available for packing (called "open" bins) are limited in number.

32.2 Online Algorithms

Let $B_1, B_2, \ldots$ denote the sequence of initially empty bins. The NEXT FIT (NF) algorithm was one of the first, and simplest, approximation algorithms to be analyzed: NF begins by packing $a_1, a_2, \ldots$ into B_1 until an item, say a_i, is encountered that does not fit, that is, $a_i > 1 - \sum_{1 \le j < i} a_j$. Item a_i is packed in B_2 and B_1 is *closed*, that is, no further items are packed in B_1. Bin B_2 becomes the new *open* bin and, like B_1, is packed with items until one that does not fit is encountered, whereupon B_2 is closed; this bin-by-bin process repeats until all items are packed. NEXT FIT is obviously linear-time; it is also bounded-space, since the number of open bins never exceeds 1.

Johnson's [2,3] parametric analysis of NF showed that

$$R^{\infty}_{NF}(\alpha) = \begin{cases} 2 & \text{if } \frac{1}{2} \le \alpha \le 1 \\ \frac{1}{1-\alpha} & \text{if } \alpha < \frac{1}{2} \end{cases}$$

As observed by Fisher [4], NF uses the same number of bins for the reverse of $L(a_n, a_{n-1}, \ldots, a_1)$, as it does for L, for every L.

An obvious drawback of NF is that it cannot make use of the empty space in closed bins. When packing a new item, the FIRST FIT (FF) algorithm tries to exploit this empty space by scanning all bins each time an item is packed; thus, bins are never closed. When packing a new item, FF puts it into the lowest indexed bin in which it fits; a new bin is started only if the current item does not fit into any nonempty bin. With an appropriate data structure, the running time of FF is $O(n \log n)$ and thus greater than the $O(n)$ running time of NF. However, FF has a much better asymptotic ratio. Johnson et al. [5] proved that, if m is a positive integer such that $1/(m+1) < \alpha \le 1/m$, then

$$R^{\infty}_{FF}(\alpha) = \begin{cases} \frac{17}{10} & \text{if } m = 1 \\ \frac{m+1}{m} & \text{if } m \ge 2 \end{cases}$$

A worst-case list proving that the 17/10 ratio cannot be improved is quite complicated. Much simpler lists showing behavior nearly as bad, in particular a 5/3 ratio, are

$$L_{6k} = \Bigg(\underbrace{\frac{1}{6} - 2\varepsilon, \ldots, \frac{1}{6} - 2\varepsilon,}_{6k} \underbrace{\frac{1}{3} + \varepsilon, \ldots, \frac{1}{3} + \varepsilon,}_{6k} \underbrace{\frac{1}{2} + \varepsilon, \ldots, \frac{1}{2} + \varepsilon}_{6k} \Bigg)$$

Then $\mathrm{OPT}(L_{6k}) = 6k$, $FF(L_{6k}) = 10k$, and the 5/3 ratio follows.

Other classical online algorithms are BEST FIT (BF), WORST FIT (WF), and ALMOST WORST FIT (AWF). BEST FIT behaves like FF, except that it puts the next item into the bin in which it fits with the smallest gap left over; ties are broken arbitrarily. WORST FIT puts the next item into a nonempty bin with the largest gap, starting a new bin only if this largest gap is too small. ALMOST WORST FIT first tries to put the next item into a nonempty bin with the second largest gap; if the item does not fit there then AWF behaves like WF. All three variants belong to the class of so-called ANY FIT (AF) algorithms: An AF algorithm scans once through the list L packing items as they are encountered. It never puts an item into an empty bin, unless it does not fit into any available partially filled bin. Similarly, an ALMOST ANY FIT (AAF) algorithm is an AF algorithm that never puts an item into a partially filled bin with the lowest level unless there is more than one bin having this level, or unless the bin of lowest level is the only one that has enough room. Johnson [3] proved that, although the class of AF algorithms is clearly large, no algorithm in this class can improve upon FF. Moreover, all AAF algorithms have the same asymptotic ratio as FF; these statements hold even for the parametric ratios. One simple consequence is that the AAF algorithms BF and AWF have asymptotic ratios 17/10. However the asymptotic ratio of the AF algorithm WF is 2, the same as that for NF.

32.2.1 Bounded-Space Online Algorithms

An online bin packing algorithm is said to use *k-bounded-space* if, for each new item, the number of bins in which it may be packed is at most k. NEXT FIT uses 1-bounded-space, whereas FF, WF, and AWF each use unbounded space. There are four very natural bounded-space bin packing algorithms that are defined via simple *packing* rules for items and simple *closing* rules for bins: A new item can always be packed into the lowest indexed bin (as in FF) or into the bin with the smallest remaining gap (as in BF). If the new item does not fit into any active bin, some active bin has to be closed; in this case one can always choose the lowest indexed bin (the FIRST bin) or the bin with the greatest sum of item sizes (the BEST bin). The corresponding four algorithms are called AFF_k, AFB_k, ABB_k, and ABF_k. Here A stands for algorithm, the second letter denotes the packing rule (best fit or first fit), the third letter denotes the closing rule (best bin or first bin), and k is the upper bound on the number of active bins. Results are as follows:

- *Algorithm AFF$_k$*. This algorithm was termed NEXT-k FIT by Johnson [3]. Provably tight bounds were not in hand until almost 20 years later. Csirik and Imreh [6] constructed a sequence of worst-case examples which show that $R^{\infty}_{AFF_k} \geq \frac{17}{10} + \frac{3}{10(k-1)}$, and Mao [7] proved a matching upper bound on $R^{\infty}_{AFF_k}$.
- *Algorithm ABF$_k$*. Mao [8] proved that $R^{\infty}_{ABF_k} = \frac{17}{10} + \frac{3}{10k}$.
- *Algorithm AFB$_k$*. Zhang [9] adapted the analysis of Mao [8] to this algorithm and proved that $R^{\infty}_{AFB_k} = \frac{17}{10} + \frac{3}{10(k-1)}$, that is, this algorithm has the same ratio as AFF$_k$.
- *Algorithm ABB$_k$*. This algorithm was investigated by Csirik and Johnson [10,11]. Compared with the other three algorithms, ABB$_k$ uses the best packing and closing rules and it has the best asymptotic ratio; $R^{\infty}_{ABB_k} = \frac{17}{10}$ holds for any $k \geq 2$.

So, except for small k, the asymptotic ratios of all four algorithms are around 17/10; since they are of the AF type, they of course cannot outperform FF.

Lee and Lee [12] introduced a new class of bounded-space algorithms using bin reservation techniques. Their algorithm HARMONIC$_k$ (H_k) is based on a partition of the interval (0, 1] into k subintervals, where the partitioning points are $1/2, 1/3, \ldots, 1/k$. To each of these subintervals there corresponds a different active bin; items belonging to a given subinterval are packed only into the corresponding active bin. If a new item arrives that does not fit into its corresponding active bin, the bin is closed and a new bin is activated. Thus, the packing in bins containing items with sizes in one of the subintervals is a NF packing. Note that H_k is not an AF algorithm because it may close a bin even when the next item could be packed in one of the active bins (belonging to a different subinterval). In Ref. [12] it was proved that

$$R^{\infty}_{H_k} \to h_{\infty} \approx 1.69103$$

This number occurs frequently in bin packing and is defined by

$$h_{\infty} = \sum_{i=1}^{\infty} \frac{1}{t_i - 1}$$

where

$$t_{i+1} = t_i(t_i - 1) + 1, \quad i \geq 1, \quad t_1 = 2 \tag{32.2}$$

Tight bounds for $R^{\infty}_{H_k}$ are not known for every value of k, so Table 32.1 gives the best upper and lower bounds currently known. The upper bounds for every value of k except 4 and 5 are from Ref. [12]. Tight upper bounds for $k \in \{4, 5\}$ are due to van Vliet, as are the lower bounds for $k \geq 4$ [13,14].

Woeginger [15] introduced the SIMPLIFIED HARMONIC (SH$_k$) algorithm, a modification of HARMONIC with a different interval structure (basically using the sequence t_i) and with a slightly better

TABLE 32.1 Asymptotic Ratios for Bounded-Space Bin Packing Algorithms, Rounded to Five Decimal Places

k	AFF$_k$	ABF$_k$	ABB$_k$	H$_k$ $\geq$	H$_k$ $\leq$	SH$_k$	Champion
2	2.00000	1.85000	1.70000	2.00000	2.00000	2.00000	ABB
3	1.85000	1.80000	1.70000	1.75000	1.75000	1.75000	ABB
4	1.80000	1.77500	1.70000	1.71429	1.71429	1.72222	ABB
5	1.77500	1.76000	1.70000	1.70000	1.70000	1.70000	ABB, H, SH
6	1.76000	1.75000	1.70000	1.70000	1.70000	1.69444	SH
7	1.75000	1.74286	1.70000	1.69444	1.69444	1.69388	SH
8	1.74286	1.73750	1.70000	1.69377	1.69388	1.69106	SH
9	1.73750	1.73333	1.70000	1.69326	1.69345	1.69104	SH
∞	1.70000	1.70000	1.70000	1.69103	1.69103	1.69103	H, SH

asymptotic ratio for small values of k. He proved that SH_k only needs $O(\log k)$ open bins to achieve the performance ratio H_k.

A summary of the asymptotic ratios of the bounded-space algorithms for some small values of k is given in Table 32.1. The asymptotic ratios of all five algorithms always remain above h_∞. In fact, Lee and Lee [12] showed that a bounded-space algorithm cannot have an asymptotic ratio better than h_∞. However, at present no online bounded-space algorithm A is known for which $R_A^\infty = h_\infty$.

Csirik and Woeginger [16] compared online bounded-space algorithms that pack into bins of size $b \geq 1$, with optimal offline algorithms that pack into bins of size 1. In a decreasing scan, choose reciprocals iteratively so long as those chosen sum to no more than $1/b$. If $1/b_i$ is the ith one chosen then $1/b = \sum_{i\geq 1} 1/b_i$, with $b_1 > b_2 > b_3 > \cdots$. The authors showed that, for every bin size $b \geq 1$, there exist online bounded-space algorithms packing into bins of size b which have a worst-case performance arbitrarily close to

$$\rho(b) := \sum_{i=1}^{\infty} \frac{1}{b_i - 1}$$

They also showed that for every $b \geq 1$, the bound cannot be beaten by any online bounded-space bin packing algorithm.

32.2.2 Better Online Algorithms

The first online algorithm for bin packing with $R_A^\infty < h_\infty$ was Yao's [17] REVISED FF (RFF). Both the definition and analysis of RFF are fairly involved. It is essentially based on FIRST FIT, but like H_k it uses separate bins for items from the intervals (0, 1/3], (1/3, 2/5], (2/5, 1/2], and (1/2, 1], respectively. Moreover, every sixth item from the interval (1/3, 2/5] is treated differently, the idea being to occasionally start a new bin with an item of this size in the hope of subsequently adding an item of size greater than 1/2 to that bin. Yao showed that

$$R_{RFF}^\infty = \frac{5}{3}$$

All other known online algorithms that beat the h_∞ bound are variants of HARMONIC that give special treatment to items $\geq 1/3$. Lee and Lee [12] described the REFINED HARMONIC (RH_K) algorithm, which was based on H_{20}. It uses the partitioning scheme of H_{20}, with the modification that the two size intervals (1/3, 1/2] and (1/2, 1] are replaced by the four intervals $(1/3, y]$, $(y, 1/2]$, $(1/2, 1-y]$, and $(1-y, 1]$, where $y = 37/96$. Packing proceeds as in a HARMONIC algorithm, except one now attempts to pair items whose sizes are in the first of the new intervals with items whose sizes are in the third interval, since such pairs can always fit in the same bin. Every seventh item with sizes in the first interval is handled differently. Lee and Lee proved that

$$R_{RH_{20}}^\infty \leq 373/228 = 1.6359\ldots$$

Ramanan et al. [18] introduced the MODIFIED HARMONIC (MH) algorithms, which added the possibility of packing still smaller items together with items having sizes in $(1/2, 1/2 + y]$, and consequently created a more complicated algorithmic structure (as well as a different value for y – in this case $y = 265/684$). For this first variant they showed that

$$1.6156146 < R_{MH_{38}}^\infty \leq 1.615615\ldots$$

and gave a general lower bound of 1.6111... for this type of algorithm. They introduced a second variant of MF (MF-2), which divides (1/3, 1/2] and (1/2, 1] into more than two parts. MF-2 has an asymptotic ratio < 1.61217 and the general lower bound 1.58333.... Hu and Kahng [19] used similar principles to construct an unnamed variant for which they claimed $R_A^\infty \approx 1.6067$.

Richey [20] introduced the HARMONIC+1 algorithm, and claimed that it has a performance ratio of 1.58872. HARMONIC+1 uses a partition of [0, 1], dividing it into more than 70 intervals. Its design and analysis were carried out with the help of linear programming.

Seiden [21] presented a general framework for analyzing a large class of online algorithms. This class includes HARMONIC, REFINED HARMONIC, MODIFIED HARMONIC, MODIFIED HARMONIC-2, and HARMONIC+1. He showed that all of these algorithms are merely instances of a general class of algorithms, which he called SUPER HARMONIC. In his new approach, he reduced the problem of analyzing an instance of SUPER HARMONIC to that of solving a specific knapsack instance. He developed a branch-and-bound algorithm for solving such knapsack problems, and furnished a general computer-assisted method of proving upper bounds for every algorithm that can be expressed in terms of SUPER HARMONIC. As a result of this new technique, Seiden found a flaw in the analysis of HARMONIC+1 and showed that the performance ratio of HARMONIC+1 is at least 1.59217. He developed a new algorithm called HARMONIC++ and showed that it has an asymptotic performance ratio of at most 1.58889. This is the current champion. He also proved that 1.58333 is a lower bound for any SUPER HARMONIC algorithm.

32.2.3 Lower Bounds for Online Algorithms

Consider next lower bounds on performance that the asymptotic worst-case ratios of *all* online bin packing algorithms must satisfy. Roughly speaking, one way to prove such lower bounds is to argue as follows. Suppose an online algorithm A is confronted initially with a huge set of tiny items. If A packs these tiny items very tightly, it would not be able to find an efficient packing for the larger items that might arrive later; if such items actually do arrive, A is going to lose. However, if A leaves lots of room for large items while packing the tiny items, the large items might not arrive; and in that case, A is again going to lose. To illustrate this idea let us consider a simple example involving two lists L_1 and L_2, each containing n identical items. The size of each item in list L_1 is $1/2 - \varepsilon$, and the size of each item in list L_2 is $1/2 + \varepsilon$. We will investigate the performance of an arbitrary online algorithm A on the following two lists: L_1 alone and the concatenation, L_1L_2, of L_1 and L_2. The items of L_1 should be packed first. The algorithm will pack some items separately (j of them, say) and it will match up the remaining $n - j$. If we stop after L_1 has been packed then, obviously, $A(L_1)/OPT(L_1) = (n + j)/n$. If L_2 comes after L_1, then the best our algorithm can do is to add one element of L_2 to each of j bins containing a single element of L_1, and to pack separately the remaining $n - j$ elements. Clearly, $A(L_1L_2)/OPT(L_1L_2) = (3n - j)/(2n)$. The best choice of j is where the maximum of the two previous ratios is minimal. It is attained when $j = n/3$, implying a lower bound of 4/3 for the asymptotic worst-case performance ratio of any online algorithm A.

Yao [17] formalized this idea using three item sizes: $1/7 + \varepsilon$ as the size of small items, and $1/3 + \varepsilon$ and $1/2 + \varepsilon$ as sizes of larger items. (Note that these item sizes are simply $1/t_i + \varepsilon$, $1 \leq i \leq 3$ as given in Eq. [32.2].) Yao proved that, given such a list, the asymptotic ratio of every online bin packing algorithm A must satisfy $R_A^\infty \geq 1.5$. Brown [22] and Liang [23] independently generalized this lower bound to 1.53635 using Yao's construction for $1 \leq i \leq 5$. Ten years later, van Vliet [13,24] found an elegant linear programming formulation for the Brown–Liang construction. Van Vliet gave an exact analysis and increased the lower bound to

$$R_A^\infty \geq 1.5401$$

This is currently the best lower bound known for online bin packing. Note that the gap between this lower bound and the upper bound of 1.58889 described in Section 32.2.2 is less than 0.05.

Galambos [25] and Galambos and Frenk [26] simplified and extended the Brown–Liang construction to the parametric case. Combining these results with the linear programming formulation of van Vliet [24] yields the lower bounds given in Table 32.2. For comparison, corresponding upper bounds for some algorithms have also been included.

Applying the above general argument, Csirik et al. [27] proved that for lists of items in nonincreasing order, no online algorithm can have an asymptotic performance ratio smaller than 8/7. Faigle et al. [28] showed that, when lists are restricted to those with only the item sizes $1/2-\epsilon$ and $1/2+\epsilon$, this lower bound becomes 4/3. Chandra [29] argued that all of these lower bounds can be carried over to *randomized* online bin packing algorithms in which the choice of an allowable packing decision can be made at random.

TABLE 32.2 Lower Bounds for the Parametric Case

α	1	1/2	1/3	1/4	1/5
Lower bound on $R_A^\infty(\alpha)$	1.540	1.389	1.291	1.229	1.188
Current champion	1.588	1.423	1.302	1.234	1.191
$R_{H_k}^\infty(\alpha)$	1.691	1.423	1.302	1.234	1.191
$R_{FF}^\infty(\alpha)$	1.700	1.500	1.333	1.250	1.200

32.3 Semionline Algorithms

Better bounds can be achieved when we relax the online condition by allowing the repacking of items. In these cases, we have to bound the number of times items are repacked, for otherwise, we would end up with an offline algorithm. Gambosi et al. [30] proposed two algorithms of this type. The first is based on a nonuniform partition of (0, 1] into four subintervals: $I_0 = (2/3, 1]$, $I_1 = (1/2, 2/3]$, $I_2 = (1/3, 1/2]$, $I_3 = (0, 1/3]$. Essentially, their first algorithm uses a HARMONIC type algorithm for each interval, but it tries to fill bins packed with items from I_1 (i.e., items with sizes in I_1) up to 2/3 with already-packed I_3 items (by repacking) or with I_3 items from the remaining part of the list. This is a linear-time algorithm (items can be packed at most twice) and it is quite easy to see that its asymptotic ratio is 3/2. Their second algorithm uses six intervals. Its time complexity is $O(n \log n)$ and the corresponding ratio is 4/3. In a subsequent paper, the same authors offered a more detailed analysis of the same set of algorithms [31].

Ivković and Lloyd [32] investigated dynamic bin packing, but as a side result they produced a quite complicated semionline algorithm with an asymptotic ratio of 5/4. They also proved a lower bound for the special class of semionline algorithms that only use *atomic* repacking moves; such a move is limited to the transfer of a single item from one bin to another. They proved that for any semionline algorithm that performs only a bounded number of atomic repacking moves at each step, the asymptotic worst-case ratio is at least 4/3 [33].

One may only be willing to consider bounded-space algorithms of the semionline type. There are such algorithms whose worst-case ratios match the limiting value, as has been shown by Galambos and Woeginger [34] and Grove [35]. The relaxation used by Ref. [34] is to allow the repacking of currently open bins, that is, items can be removed from current open bins and reassigned before packing the current item. Galambos and Woeginger presented an "online with repacking" algorithm REP$_3$ that uses a weighting function w and three open bins concurrently as follows. The current item is always packed in an open bin. Then all elements in the three open bins are repacked by the FIRST FIT DECREASING (FFD) algorithm, with the result that either one bin becomes empty or at least one bin B has $w(B) := \sum_{a_i \in B} w(a_i) \geq 1$. All bins with $w(B) \geq 1$ are then closed and replaced by empty bins. The authors proved that

$$R_{REP_3}^\infty = h_\infty$$

Grove [35] independently designed an algorithm with the same behavior using an alternative notion that he calls *lookahead*. In his algorithm, one is given a fixed *warehouse size, W*; an item a_i need not be packed until one has looked at every item a_i through a_j, for $j > i$ such that $\sum_{h=i}^{j} s(a_h) \leq W$. Allowing lookahead, an appropriate choice of parameters again gives an algorithm with an asymptotic worst-case ratio of h_∞.

32.4 Offline Algorithms

As unconstrained algorithms for partitioning the sets $\{a_i\}$, offline algorithms need not be thought of in terms of sequential packings of an ordered list. However, the best known offline algorithms are usually expressed in just these terms, but with an initial ordering of the list allowed. That this is a natural first approach can be seen from the bad examples for online algorithms, which were either increasing (FF) or alternating (NF) sequences of item sizes. Thus, sorting the items in decreasing order and then employing

NF, FF, or BF creates three interesting candidates for simple, but effective offline algorithms; they are denoted by NFD, FFD, and BFD, with the "D" standing for "Decreasing." The sorting needs $O(n \log n)$ time and so the total running time of each algorithm will be $O(n \log n)$.

Baker and Coffman [36] proved that $R^{\infty}_{NFD} = h_{\infty}$, and they gave a parametric bound as well: if $\alpha \in (1/(s+1), 1/s]$, $s \geq 1$ then

$$R^{\infty}_{NFD}(\alpha) = h_{\infty}(\gamma_s^*)$$

where $h_{\infty}(\gamma_s^*) = \frac{s-1}{s} + \gamma_s$. Here γ_s is a slight generalization of the previous t_i sequence, as follows: $t_1(s) = s+1$, $t_2(s) = s+2$, and $t_{i+1}(s) = t_i(s)(t_i(s)-1)+1$, for $i \geq 2$. Then $\gamma_s = \sum_{i=1}^{\infty} \frac{1}{t_i(s)-1}$.

As expected, better results are achieved when using FF (or BF) after the initial sort. Johnson [2] showed that for every list L

$$FFD(L) \leq \frac{11}{9} \cdot OPT(L) + 4$$

He could also prove that this bound is tight, that is, $R^{\infty}_{FFD} = 11/9$. Later, Baker [37] gave a shorter and simpler proof and cut the constant to 3. Further shortening in the proof and reducing the constant to 1 was given by Yue [38]. Later, even this value was cut to 7/9 by Li and Yue [39].

For the parametric case, Johnson [2] showed that

$$R^{\infty}_{FFD}(\alpha) = \begin{cases} \frac{71}{60} & \text{for } \frac{8}{29} < \alpha \leq \frac{1}{2} \\ \frac{7}{6} & \text{for } \frac{1}{4} < \alpha \leq \frac{8}{29} \\ \frac{23}{20} & \text{for } \frac{1}{5} < \alpha \leq \frac{1}{4} \end{cases}$$

and he conjectured that

$$R^{\infty}_{FFD}\left(\frac{1}{m}\right) = 1 + \frac{1}{m+2} - \frac{2}{m(m+1)(m+2)} = F_m$$

for all integers $m \geq 4$.

Csirik [40] proved that this conjecture is true when m is even but is false when m is odd. Defining

$$G_m = 1 + \frac{1}{m+2} - \frac{1}{m(m+1)(m+2)} = F_m + \frac{1}{m(m+1)(m+2)}$$

he was able to prove that

$$R^{\infty}_{FFD}\left(\frac{1}{m}\right) = \begin{cases} F_m & \text{if } m \text{ is even} \\ G_m & \text{if } m \text{ is odd} \end{cases}$$

for all $m \geq 5$.

Xu [41] completed the proof for arbitrary values of $\alpha \leq 1/4$. He showed that when m is even, F_m holds true for any $\alpha \in (1/(m+1), 1/m]$, while for m odd the interval has to be divided into two parts with

$$R^{\infty}_{FFD}(\alpha) = \begin{cases} F_m & \text{if } \frac{1}{m+1} < \alpha \leq d_m \\ G_m & \text{if } d_m < \alpha \leq \frac{1}{m} \end{cases}$$

where $d_m := (m+1)^2/(m^3 + 3m^2 + m + 1)$.

The lower bound for FFD is given via the following lists:

$$L_{6k} = \Bigg(\underbrace{\frac{1}{2}+\varepsilon, \ldots, \frac{1}{2}+\varepsilon}_{6k}, \underbrace{\frac{1}{4}+2\varepsilon, \ldots, \frac{1}{4}+2\varepsilon}_{6k}, \underbrace{\frac{1}{4}+\varepsilon, \ldots, \frac{1}{4}+\varepsilon}_{6k}, \underbrace{\frac{1}{4}-2\varepsilon, \ldots, \frac{1}{4}-2\varepsilon}_{12k}\Bigg)$$

It is then straightforward to check that $\text{OPT}(L_{6k}) = 9k$ and $FFD(L_{6k}) = 11k$.

After sorting the items we may use an AF algorithm to pack the list. Unfortunately, there are no exact bounds for this case. We only know [2,3,5] that

$$\frac{11}{9} \le R^{\infty}_{AF}(1) \le \frac{5}{4}$$

and

$$\frac{1}{m+2} - \frac{2}{m(m+1)(m+2)} \le R^{\infty}_{AF}(\alpha) \le \frac{1}{m+2}$$

where $m = \left\lfloor \frac{1}{\alpha} \right\rfloor$ and $\alpha < 1$.

Johnson [2] made an interesting attempt to get a better offline algorithm. He proposed the MOST-k FIT (MF$_k$) algorithm, which first sorts the list in decreasing order. The next bin to be packed will first be filled with the largest as yet unpacked element. If the smallest item in the list does not fit in the bin, we close the bin and continue with the next bin. If the smallest item does fit, we pack at most k additional items with the least space left over. The running time is $O(n^k \log n)$. For a long time, Johnson conjectured that $\lim_{k\to\infty} R^{\infty}_{MF_k} = 10/9$, but Friesen and Langston [42] provided a counterexample showing that $R^{\infty}_{MF_k} \ge 5/4$ for $k \ge 2$.

For several years FFD had the provably smallest asymptotic bound. Yao [17] proposed a complicated algorithm A with a $O(n^{10} \log n)$ running time and with $R^{\infty}_{A} \le 11/9 - 10^{-7}$, which proved that FFD could in fact be beaten. A much larger improvement was achieved by Garey and Johnson [43]. They proposed the MODIFIED FIRST FIT DECREASING (MFFD), which differs from FFD only when packing the items from $(1/6, 1/3]$, called *key* items, just after packing all items $> 1/3$. MFFD attempts to pack 2 key items in every *key* bin, defined as a bin having only an item $> 1/2$. The key bins are packed in largest-gap-first order. If the gap in the current key bin can accommodate the two smallest items in the set S_U of the as yet unpacked key items, then the first key item packed is the smallest in S_U and the second one packed is the one remaining in S_U that would be chosen by BF. As soon as at most one key item remains or a key bin is encountered that cannot accommodate the smallest two remaining key items, MFFD reverts to FFD for the remainder of the packing. Garey and Johnson proved that

$$R^{\infty}_{MFFD} = \frac{71}{60} = 1.183333\ldots.$$

This is the best known offline algorithm. It is interesting to note that $R^{\infty}_{MFFD} = R^{\infty}_{FFD}\left(\frac{1}{2}\right)$.

In their attempt to find a better algorithm, Friesen and Langston [42] investigated the BEST TWO FIT (B2F) algorithm, which is basically a MF$_k$-type algorithm with $k = 2$: B2F first pack a bin by FFD. If the bin contains more than a single item, then the list is checked to see if the smallest item in the bin could be replaced by two items that would occupy more of the bin. If so, the two largest such items are inserted in place of the smallest item. The algorithm is also simplified by requiring all items smaller than 1/6 be withheld until all larger items have been packed. FFD is used to complete the packing when only items no greater than 1/6 are left. The authors proved that

$$R^{\infty}_{B2F} = \frac{5}{4} = 1.25$$

which is worse than those of FFD. The bad lists for this algorithm have the following optimal packing:

$$\left(\underbrace{\frac{1}{3} - 4^k\frac{\varepsilon}{2}, \frac{1}{3} - 4^k\frac{\varepsilon}{2}, \frac{1}{3} + 4^k\varepsilon,}_{2\text{ bins}} \underbrace{\frac{1}{3} - 4^{k-1}\frac{\varepsilon}{2}, \frac{1}{3} - 4^{k-1}\frac{\varepsilon}{2}, \frac{1}{3} + 4^{k-1}\varepsilon,}_{8\text{ bins}} \ldots, \right.$$

$$\left. \underbrace{\frac{1}{3} - 2\varepsilon, \frac{1}{3} - 2\varepsilon, \frac{1}{3} + 4\varepsilon,}_{2\cdot 4^{k-2}\text{ bins}} \underbrace{\frac{1}{3} + \varepsilon, \frac{1}{3} + \varepsilon}_{4^{k-1}\text{ bins}} \right)$$

Fortunately, the counterexamples for FFD and B2F are complementary and so the authors defined a compound algorithm (CFB) that runs both FFD and B2F on the list and takes the better result. The running time may double, but for the performance ratio we get

$$1.164\ldots = \frac{227}{195} \le R^{\infty}_{CFB} \le \frac{6}{5} = 1.2$$

The worst-case example is given by an example where the bin size is 559, and lists where the optimal packing consists of $60k$ bins each having three items of size 381, 98, 80, of $120k$ bins each having five items of size 191, 96, 96, 96, 80, and of $15k$ bins each having six items of size 99, 99, 99, 99, 80, 80. Then the optimal packing clearly uses $195k$ bins, and CFB will use $227k$ bins. The exact bound for CFB has not yet been determined and so it is not known whether it is better than MFFD.

An earlier step in this direction (i.e., sequentially packing each bin as well as possible) did not entail orderings by size. Graham [44] proposed a greedy algorithm in which, at each step, packing the next bin solved a knapsack problem. Surprisingly, however, the worst-case behavior of this algorithm A is rather poor:

$$R^{\infty}_{A} \ge \sum_{k=1}^{\infty} \frac{1}{2^k - 1} = 1.6067\ldots$$

This lower bound is proved by instances L_b that have b items of size $1/2^k + \varepsilon$ for $k = 1, 2, \ldots, p$, with $0 < \varepsilon \le 1/2^{2p}$ and for values of b that are multiples of $2^k - 1$, $k = 1, 2, \ldots, p$, for example, $b = a\prod_{k=1}^{p}(2^k - 1)$ for some positive integer a. Then the optimal packing of L_b has b bins, each bin consisting of one item from each size. Grahams's algorithm first packs $b/(2^p - 1)$ bins, each with $2^p - 1$ items of size $1/2^p + \varepsilon$, then $b/(2^{p-1} - 1)$ bins, each with $2^{p-1} - 1$ items of size $1/2^{p-1} + \varepsilon$, and so on, ending with b bins, each with one item of size $1/2 + \varepsilon$. Summing up, the lengths of the subpackings gives the lower bound.

Caprara and Pferschy [45] recently proved an upper bound on the asymptotic performance of Graham's algorithm:

$$R^{\infty}_{SS} \le \frac{4}{3} + \ln\left(\frac{4}{3}\right) \approx 1.6210$$

They also dealt with the parametric case, showing that, if $\frac{1}{m+1} < \alpha \le \frac{1}{m}$, then

$$R^{\infty}_{SS}(\alpha) \ge \sum_{k=1}^{\infty} \frac{m}{(m+1)^k - 1}$$

and

$$R^{\infty}_{SS}(\alpha) \le \begin{cases} 2 - \frac{4m}{3(m+1)} + \ln\left(\frac{4}{3}\right) & \text{if } m \le 2 \\ 1 + \ln\frac{m+1}{m} & \text{if } m \ge 3 \end{cases}$$

32.5 Other Worst-Case Issues

32.5.1 Special Case Optimality

One of the easiest cases is when the number of different items in the lists is bounded, that is, we have k different item sizes. Let us denote the item sizes by $s_1, s_2, \ldots, s_k$. In this case we have a smallest item size. Then let us denote the smallest integer by j so that all item sizes are $\ge 1/j$. It is quite clear that we cannot pack more than j items in one bin. So a legal packing (we call it here a configuration) of a bin can be given by a k-tuple,

$$p_i = (p_{i1}, p_{i2}, \ldots, p_{ik}), \sum_{l=1}^{k} p_{il} \le j, \sum_{l=1}^{k} s_l \cdot p_{il} \le 1$$

where p_{ij} denotes the number of s_j items in configuration i. The total number of lists having not more than j items is $O(k^j)$, a part of them form a configuration. But then the total number of all possible packings of a list $L = (a_1, a_2, \dots, a_n)$ where $a_i \in \{s_1, s_2, \dots, s_k\}$ is $O(n^{k^j})$, that is, a polynomial in n. This is a brute force method—the optimum can be computed more easily in this case, as proposed by Blazewicz and Ecker [46]. Let us denote the number of items s_i in the list L by m_i, $1 \le i \le k$. Then the problem of finding the optimal packing of L can be formulated as the following integer programming problem:

$$\min \sum_{i=1}^{N} b_i, \quad w.r.t \sum_{i=1}^{N} p_{ij} b_i = m_j \quad (j = 1, \dots, k)$$

and $b_i \ge 0$ are integers $(i = 1, \dots, N)$. Here N is the number of all possible configurations and b_i gives the number of bins of configuration p_i. This matrix has k rows and $O(k^j)$ columns and can be solved in polynomial time in the number of constraints [47]. The optimal packing can then be constructed in linear time. If we do not need the exact optimum but just a good approximation of it then we can relax the integer condition from the above inequalities and compute the optimum of the remaining problem. Then we may round down the solution, as proposed by Gilmore and Gomory [48,49].

A different approach was chosen by Hochbaum and Schmoys [50]. They investigated the classical bin packing problem with item sizes bounded from below. They found an easily solvable case: if all item sizes are $\ge 1/3$, then a relatively straightforward polynomial algorithm will furnish an optimal packing. This algorithm will pack, in the first stage, all items larger than a half separately in bins. In the second stage all items $= 1/3$ are packed by three items in bins. In the third stage the remaining items, that is, those with size $> 1/3$ and $\le 1/2$ are paired up with items from stage one, and those items where the pairing is not possible are packed by two. The second step in this direction was to prove that if item sizes are $\ge 1/3 - \varepsilon$ for any fixed ε, $0 < \varepsilon < 1/3$, then the problem becomes strongly NP-complete. Even for this case, supposing $0 < \varepsilon \le 1/12$, and slightly relaxing the problem allowing the bins to be overpacked a little bit, they gave a polynomial-time algorithm that gives the optimal number of bins but an overpacking with at most $3\varepsilon/2$.

Leung [51] derived the limits for polynomially solvable cases. He proved that the bin packing problem remains NP-hard even if the bin capacity is one and if item sizes must be drawn from the set $\{1, r, r^2, r^3, \cdots\}$, where r is an arbitrary positive rational number < 1.

Another type of optimality result was covered by Coffman et al. [52]. They introduced the restricted lists L where the item sizes form *divisible sequences.* Now let us suppose that the bin capacity is an integer. A sequence of item sizes $s_1 > s_2 > \dots > s_i > s_{i+1} > \dots$ is called *divisible* if for each $i \ge 1$, s_{i+1} exactly divides s_i. A list L is called *weakly divisible* if its items when sorted forms a divisible sequence. If, in addition, the largest item size exactly divides the bin capacity, then the list is called *strongly divisible.* In Ref. [52] it was proven that when L is weakly divisible FFD gives optimal packing. If L is strongly divisible then even FF produces optimal packing.

32.5.2 Linear-Time Algorithms

Most algorithms investigated up till now have a running time superlinear in n. It would be interesting to know what could be said about algorithms that have a linear running time. Naturally, NF is one of these, but it has poor performance. Johnson quite early on [3] proposed a better linear-time algorithm, GROUP-X FIT GROUPED (GXFG). First he defined the GROUP-X FIT (GXF) algorithm. Let a *schedule of intervals* be the set $X = \{x_0, x_1, \dots, x_k\}$, where $x_0 = 0$, $x_k = 1$, and $x_i < x_{i+1}$, $0 \le i < k$. X can be thought of as a partition of the unit interval $[0, 1]$ into the subintervals $(0, x_1]$, $(x_1, x_2]$, $\dots$, $(x_{k-1}, 1]$. Interval $(x_{i-1}, x_i]$ will be called the X_i, $1 \le i \le k$ interval, and the points in X will be called breakpoints. Items from the list are classified similarly: a_j will be called an I_l item if it belongs to X_l. Note that this is a more general definition than that given for the Harmonic algorithm, the latter having $x_i = 1/i$ as breakpoints. GROUP-XFIT will round down the free space in open bins to the next breakpoint (if the free space is equal to a breakpoint then it is kept). The algorithm is actually a variant of best fit: an I_l item will be packed in a bin with the least free space left over (if no bin has enough space for the item, a new bin will be used). The algorithm has linear running time and is online. Johnson [2,3] did show that for $m = \lfloor 1/\alpha \rfloor \geq 2$,

$$R^{\infty}_{GXF}(\alpha) = R^{\infty}_{FF}(\alpha)$$

when $\{1/(m+1), 1\} \subseteq X$. He also proved that if $m = 1$, then $1.7 \leq R^{\infty}_{GXF} \leq 2$ whenever $\{1/6, 1/3, 1/2\} \subseteq X$ and conjectured that the lower bound is the tight value.

The better algorithm GXFG first scans through the list and collects items from the same intervals together. The second step is just GXF, which starts with items from interval X_k, followed by items from X_{k-1}, X_{k-2}, ... Johnson also proved, for all $m = \lfloor 1/\alpha \rfloor \geq 1$, if X contains $1/(m+2)$, $1/(m+1)$, $1/m$, that

$$R^{\infty}_{GXFG}(\alpha) = \frac{m+2}{m+1}$$

Because of the first scan GXFG is not an online algorithm. Basically, it uses the idea of FFD (it first starts to pack the large elements) but without really sorting the elements.

A further improvement was achieved by Martel [53]. He used subintervals too, but packed items more freely than Johnson in his GXFG. The algorithm is the following. First, we partition the items using the following five sets:

$$C_1 = \{x_i \mid 2/3 < x_i \leq 1\}$$
$$C_2 = \{x_i \mid 1/2 < x_i \leq 2/3\}$$
$$C_3 = \{x_i \mid 1/3 < x_i \leq 1/2\}$$
$$C_4 = \{x_i \mid 1/4 < x_i \leq 1/3\}$$
$$C_5 = \{x_i \mid x_i \leq 1/4\}$$

Let $c_i = \mid C_i \mid$, $i = 1, 2, \ldots, 5$, be the numbers of items in intervals. An item in C_i will be called a C_i-piece. The motivation behind this partitioning is to allow items to be packed based on the set to which they belong. Items in C_1 can only be combined with items in C_5. Items in C_2 can always be combined with an item in C_4, but never with an item in C_2. Items in C_3 can be packed together two to a bin, items in C_4 can be packed three to a bin, and items in C_5 can be packed at least four to a bin. The algorithm OffBP is based on these observations and is defined in the following:

Step 1. Form the sets C_i, $i = 1, 2, \ldots, 5$.

Step 2. Let $k = \lceil \min\{c_2, c_3\}/2 \rceil$. Split C_2 into the two sets C_2^s that contains the smallest k elements in C_2 and C_2^b, which contains the remaining elements. In a similar way, split C_3 into C_3^s and C_3^b. Pair up each element in C_2^s with an element in C_3^s. If the pair fits in a bin, create a bin containing both items. If the pair does not fit, create a bin containing only the element from C_2^s.

Step 3. Put each C_1-piece into its own bin and put each C_2^b-piece into its own bin.

Step 4. Put all C_3^b-pieces and all C_3^s-pieces which were not combined with a C_2^s-piece in step 2 into bins, two into each bin.

Step 5. Each bin with a single C_2-piece has a C_4-piece added to it (until one runs out of C_4-pieces).

Step 6. Any C_4-pieces not packed in step 5 are put three to a bin into empty bins.

Step 7. Pack C_5-pieces into the bins created in steps 1–6 using NF: for each bin we add C_5-pieces until the next C_5-piece does not fit; then go to the next bin and never consider the previous bin again.

Step 8. Any remaining C_5-pieces are packed into empty bins using NF.

Martel has shown that the performance ratio of OffBP is 4/3. He also gave hints on how the algorithm could perhaps be improved. These were followed up by Békési et al. [54]. They used the breakpoints {1/5, 1/4, 1/3, 3/8, 1/2, 2/3, 4/5} and a much more complicated algorithm and obtained a performance bound of 5/4.

32.5.3 Randomization

Bin packing was a pioneering theme in many fields of analysis of algorithms. This is the case for randomized algorithms as well. Perhaps AF is the first randomized algorithm that was really analyzed in detail. This is certainly a classical result for randomized algorithms.

Only a few additional results are known for randomized algorithms in bin packing. As mentioned earlier, Chandra [29] proved that the general lower bound for online algorithms hold for randomized algorithms too.

Kenyon [55] analyzed BF from a new point of view and defined a new measure of performance. This new measure is the expectation over random packing orders of a *worst-case list* of input items; packing orders are drawn uniformly at random from the set of all permutations. She defined the random-order performance-ratio RC(A) of an online algorithm A as

$$RC(A) = \limsup_{OPT(L)\to\infty} \frac{E_\sigma A(L_\sigma)}{OPT(L)}$$

where L_σ is the permuted list $(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ and the summation for the expectation is taken over all permutations $\sigma \in \mathcal{S}_n$. She proved that for BF

$$1.08 \leq RC(BF) \leq 1.5$$

a quite large gap between the lower and upper bound! She also conjectured that the true answer was probably close to 1.15. To estimate the lower bound she used a nice simplification: if we have only two different item sizes then the permutations can be simulated by drawing each item independently with the appropriate probabilities. So the sequence can be viewed as an unbiased random walk in the plane, which can be analyzed with help of the Markov chains. The lower bound was found with the aid of the lists where we have 1/2 items with probability p and 1/3 items with probability $1 - p$. In this case we have a Markov chain with five states:

(a) There is no open bin.
(b) There is one open bin whose current content is 1/2.
(c) There is one open bin whose current content is 1/3.
(d) There is one open bin whose current content is 2/3.
(e) There are two open bins, one containing 1/2 and the other containing 2/3.

A quite standard analysis of this Markov chain leads to the lower bound 1.08. The proof of the upper bound is much more complicated and uses some deeper results from probability theory.

No further results are available for this type of investigation. We are not aware of any estimates for $RC(NF)$!

32.5.4 Absolute Worst-Case Ratios

Up till now we have considered the asymptotic worst-case ratios of bin packing heuristics. For lists where OPT(L) is small, a different ratio may also be helpful. This is the absolute worst-case ratio, which is defined by Eq. (32.1). It can readily be seen that bin packing is hard even for lists for which $OPT(L)$ is small. This is true because the 2-partition problem, which is known to be NP-complete [56], can be polynomially reduced to the problem whether or not it is possible to pack all items in two bins. This also implies that no polynomial-time heuristic has an absolute worst-case ratio smaller than 1.5 for the bin packing problem, unless $P = NP$. This is obvious as such a heuristic could solve the 2-partition in polynomial time [56]. However, for the list $L = (0.4, 0.4, 0.3, 0.3, 0.3, 0.3)$, $OPT(L)) = 2$ and $FFD(L) = 3$, so FFD has the best possible absolute worst-case bound. From Ref. [39] it is known that

$$FFD(L) \leq \frac{11}{9} \cdot OPT(L) + \frac{7}{9}$$

for all lists. This gives an upper bound of 4 for lists with OPT(L)=3 and bound of 5 for lists with OPT(L)=4. The same numbers apply to BF too.

As regards FF (and BF), we have already mentioned [57] that, for any list L,

$$FF(L) \le \left\lceil \frac{17}{10} \cdot OPT(L) \right\rceil$$

and Ref. [5] gave examples showing that $FF(L)/OPT(L) = 17/10$ could be actually attained. Simchi-Levi [58] once proved that for all lists $FF(L) \le 1.75 \cdot OPT(L)$.

Xie and Liu [59] improved the upper bound, giving a proof for

$$FF(L) \le \min\left(\left\lfloor \frac{33}{19} \cdot \mathrm{OPT}(L) \right\rfloor , \left\lceil \frac{17}{10} \cdot \mathrm{OPT}(L) \right\rceil \right)$$

Summarizing the results we currently know that

$$1.7 \le FF(L) \le \min\left(\left\lfloor \frac{33}{19} \cdot \mathrm{OPT}(L) \right\rfloor , \left\lceil \frac{17}{10} \cdot \mathrm{OPT}(L) \right\rceil \right)$$

Berghammer and Reuter [60] gave a linear-time algorithm with an absolute worst-case bound of 3/2. This bound holds for FFD and BFD too, but in $O(n \log n)$ time. The algorithm is based on the BF idea. It works with two partial solutions P_1 and P_2 and two auxiliary bins B_1 and B_2—one for each partial solution. The algorithm proceeds as follows: First, the items are packed one by one into B_1 until its capacity would be exceeded by the packing of some item u. In this situation the contents of B_1 is inserted into P_1, the bin B_1 is cleared, u is packed into B_2, and the process is repeated with the remaining items. If, however, the packing of u would lead to an overfilling of B_1 as well as B_2 then, in addition, the contents of B_2 is inserted into P_2 and u is packed into the cleared B_2. This bookkeeping combined with a suitable selection of the next object (based on a partition of the objects into small and large ones at the beginning of the algorithm) allows us to avoid the costly search of a bin the next item will optimally fit in.

Zhang et al. [61] furnished a different offline algorithm with absolute worst-case ratio 3/2 using an *extra* bin. The algorithm is the following:

Step 1. Put large items (>1/2) each into a bin. Index the unfilled bins in arbitrary order. Set all bins as active bins. Then repeatedly arrange small items as follows.

Step 2. If there is an active open bin, put the current item a_i into the open active bin with the lowest index if the bin has enough room for a_i, otherwise close this active bin and consider the extra bin as follows. If there is an extra bin open and it has enough room for a_i, place a_i into it. Otherwise close the extra bin. Open a new bin for a_i and set this bin as the extra bin.

Step 3. If there is no active bin open, create a new bin for a_i and set this new bin as the active bin.

Actually, in step 1, we need to only arrange one large item at a time. As soon as the active bin containing a large item is closed, we open a new bin and put the remaining large item into it first and set it as the active bin. At any one time we keep at most one active bin and at most one extra bin open. So it is a bounded-space algorithm. The authors proved that its tight absolute worst-case ratio is 3/2.

They also defined an online algorithm of the same type. This is again a bounded-space algorithm and has a tight absolute worst-case ratio of 7/4. In this algorithm, a bin is called an L-bin if the first item it accepts is a large item and has not been set as an active bin. Once such bin becomes an active bin it will no longer be called an L-bin. The algorithm is the following:

Step 1. Open a bin for the first item. Set it as the active bin.

Step 2. If there is an active bin B_j open, put the current item a_i into B_j if it has enough room for this item. But suppose that B_j does not have enough space for the item:

(a) If a_i is a large item, create a new bin for it. If there are two L-bins, close them.

(b) If a_i is a small item, close the active bin. Set an open L-bin as the active bin if such a bin exists. If there is an extra bin open and this bin has enough room for a_i, pack a_i into it. Otherwise open a new bin for a_i and set the new bin as the extra bin.

Step 3. If there is no active bin open, create a new bin for item a_i and set it as the active bin.

32.5.5 Bounds on Anomalous Behavior

As we have already seen, the two well-known algorithms FF and BF have the same asymptotic ratios. However they can give strikingly different packings for individual lists. Examples are given in Ref. [2] of lists L with arbitrarily large values of OPT(L), both such that BF(L) = (4/3)·FF(L) and FF(L) = (3/2)·BF(L). Similarly, AWF can be just as far away from FF, and examples exist such that AWF(L) = (5/4)·FF(L) and that FF(L) = (9/8)·AWF(L).

The decreasing-type algorithms FFD and BFD may be likewise compared. In Ref. [2] lists are given for which BFD(L) = (10/9)·FFD(L). However, BFD can produce better packings than FFD as well: There are lists for which FFD(L) = (11/10)·BFD(L).

It has been shown that certain algorithms possess the undesirable property of sometimes performing "worse" when their inputs are made "better." Let us start from a list L and let us form a new list L' by deleting some elements of L and/or reducing the size of some elements of L. We will say in this case that *L dominates L'*. If an algorithm never uses more bins to pack L' than it uses to pack L we will say that the algorithm is monotonic, otherwise we will say the algorithm is nonmonotonic. Graham [44] and Johnson [2] once showed that FF is nonmonotonic. Murgolo [62] introduced a technique that allows one to completely characterize the monotonic behavior of any algorithm in a large, natural class. He provided upper bounds on nonmonotonic behavior for any *reasonable* algorithm. (A reasonable algorithm is one which never packs an item into an empty bin if it can fit into an already allocated bin.) We note that a reasonable algorithm is an AF algorithm. For example, he showed that there exist arbitrarily long lists satisfying *L dominates L'* such that $FF(L') \geq FF(L) + \frac{1}{75}FF(L)$, and this bound is tight to within a constant factor. Similar results hold for BF (with a constant of $\frac{1}{42}$) and WF (with a constant of $\frac{1}{15}$).

32.5.6 Parallel Algorithms

Anderson, Mayr, and Warmuth have investigated the FFD algorithm from a parallel computational view. They used the PARALLEL RANDOM ACCESS MACHINE (PRAM) model of parallel computation of Fortune and Wyllie [63]. They considered a parallel algorithm to be fast if it is an NICK'S CLASS ($\mathcal{N}C$) algorithm, that is, if it runs in polylogarithmic time using a polynomial number of processors. The main algorithm they gave has a more reasonable bound, running in $O(\log n)$ time on an $n/\log n$ processor EREW (exclusive read, exclusive write) PRAM, and so is asymptotically optimal. A problem is called inherently sequential if it is $\mathcal{P}$-complete. This furnishes a relatively strong evidence that the problem is not in $\mathcal{N}C$, since if it were, we would have $\mathcal{P} = NC$. The interpretation of this is that deciding the value of a specific bit of the output of the algorithm is $\mathcal{P}$-complete.

The main result of Ref. [63] is that the FFD heuristic is a $\mathcal{P}$-complete algorithm, and that a packing which obeys the same performance bound as FFD can be computed by a fast parallel algorithm. The algorithm packs the large items (items of size $\geq 1/6$) in the same manner as FFD and then fills in the remaining items. Now let $u_1, u_2, \ldots, u_r$ be the list of remaining items, all with a size less than 1/6. We first combine these items into chunks so that every chunk (except possibly the last) has a size between 1/24 and 1/6. The items with a size 1/24 or larger are big enough and each is put into a chunk by itself. For the remaining items, the partial sums $s_k = \sum_{1<j\leq k} u_j$ are determined and for each i, we combine the set of items $\{u_k \mid i/12 \leq s_k < (i+1)/12\}$ to form a chunk. Since the item sizes are less than 1/24, each chunk will have a size between 1/24 and 3/24.

Now the bins packed using the FFD algorithm with items of size 1/6 or larger will be filled in. We will add, in parallel, a distinct chunk to each bin filled less than 5/6. Since the sizes are at least 1/24, only a constant number of passes are needed. If there are any leftover items they are packed in new bins, just as in the method used in FFD packing.

The algorithm runs in $O(\log n)$ time using $n/\log n$ processors.

References

[1] Garey, M. R., Graham, R. L., and Ullman, J. D., Worst-case analysis of memory allocation algorithms, *Proc. STOC*, ACM, New York, 1972, p. 143.

[2] Johnson, D. S., Near-Optimal Bin Packing Algorithms, Ph.D. thesis, Massachusetts Institute of Technology, Department of Mathematics, Cambridge, 1973.

[3] Johnson, D. S., Fast algorithms for bin packing, *JCSS*, 8, 272, 1974.

[4] Fisher, D. C., Next-fit packs a list and its reverse into the same number of bins, *Oper. Res. Lett.*, 7, 291, 1988.

[5] Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., and Graham, R. L., Worst-case performance bounds for simple one-dimensional packing algorithms, *SIAM J. Comput.*, 3, 299, 1974.

[6] Csirik, J. and Imreh, B., On the worst-case performance of the NkF bin-packing heuristic, *Acta Cybern.*, 9, 89, 1989.

[7] Mao, W., Tight worst-case performance bounds for next-k-fit bin packing, *SIAM J. Comput.*, 22, 46, 1993.

[8] Mao, W., Best-k-fit bin packing, *Computing*, 50, 265, 1993.

[9] Zhang, G., Tight Worst-Case Performance Bound for AFB$_k$ Bin Packing, Technical report 015, Institute of Applied Mathematics, Academia Sinica, Beijing, China, May 1994.

[10] Csirik, J. and Johnson, D. S., Bounded space on-line bin packing: best is better than first, *Proc. SODA*, SIAM, 1991, p. 309.

[11] Csirik, J. and Johnson, D. S., Bounded space on-line bin packing: best is better than first, *Algorithmica*, 31, 115, 2001.

[12] Lee, C. C. and Lee, D. T., A simple on-line packing algorithm, *JACM*, 32, 562, 1985.

[13] van Vliet, A., Lower and Upper Bounds for On-Line Bin Packing and Scheduling Heuristic, Ph.D. thesis, Erasmus University, Rotterdam, The Netherlands, 1995.

[14] van Vliet, A., On the asymptotic worst case behavior of harmonic fit, *JACM*, 20, 113, 1996.

[15] Woeginger, G. J., Improved space for bounded-space, on-line bin-packing, *SIAM J. Disc. Math.*, 6, 575, 1993.

[16] Csirik, J. and Woeginger, G. J., Resource augmentation for online bounded space bin packing, in *ICALP*, Montanari, U., Rolim, J. D. P., and Welzl, E., Eds., Lecture Notes in Computer Science, Vol. 1853, Springer, Berlin, 2000, p. 296.

[17] Yao, A. C., New algorithms for bin packing, *JACM*, 27, 207, 1980.

[18] Ramanan, P., Brown, D. J., Lee, C. C., and Lee, D. T., On-line bin packing in linear time, *J. Algorithms*, 10, 305, 1989.

[19] Hu, T. C. and Kahng, A. B., Anatomy of On-Line Bin Packing, Technical report CSE-137, Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, 1988.

[20] Richey, M. B., Improved bounds for harmonic-based bin packing algorithms, *Disc. Appl. Math.*, 34, 203, 1991.

[21] Seiden, S. S., On the online bin packing problem, *JACM*, 49, 640, 2002.

[22] Brown, D. J., A Lower Bound for On-Line One-Dimensional Bin Packing Algorithms, Technical report 864, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1979.

[23] Liang, F. M., A lower bound for on-line bin packing, *Inf. Proc. Lett.*, 10, 76, 1980.

[24] van Vliet, A., An improved lower bound for on-line bin packing algorithms, *Inf. Proc. Lett.*, 43, 277, 1992.

[25] Galambos, G., Parametric lower bound for on-line bin-packing, *SIAM J. Alg. Disc. Meth.*, 7, 362, 1986.

[26] Galambos, G. and Frenk, J. B. G., A simple proof of Liang's lower bound for on-line bin packing and the extension to the parametric case, *Disc. Appl. Math.*, 41, 173, 1993.

[27] Csirik, J., Galambos, G., and Turán, Gy., Some results on bin-packing, *Proc. of EURO VI*, Vienna, 1983, p. 52.

[28] Faigle, U., Kern, W., and Turán, Gy., On the performance of on-line algorithms for partition problems, *Acta Cybern.*, 9, 107, 1989.

[29] Chandra, B., Does randomization help in on-line bin packing? *Inf. Proc. Lett.*, 43, 15, 1992.

[30] Gambosi, G., Postiglione, A., and Talamo, M., New algorithms for on-line bin packing, in *Algorithms and Complexity, Proc. of the 1st Italian Conf.*, Ausiello, G., Bovet, D., and Petreschi, R., Eds., World Scientific, Hackensack, NJ, 1990, p. 44.

[31] Gambosi, G., Postiglione, A., and Talamo, M., Algorithms for the relaxed online bin-packing model, *SIAM J. Comput.*, 30, 1532, 2000.

[32] Ivković, Z. and Lloyd, E., Fully dynamic algorithms for bin packing: being (mostly) myopic helps, *SIAM J. Comput.*, 28, 574, 1998.

[33] Ivković, Z. and Lloyd, E., A fundamental restriction on fully dynamic maintenance of bin packing, *Inf. Proc. Lett.*, 59, 229, 1996.

[34] Galambos, G. and Woeginger, G. J., An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling, *SIAM J. Comput.*, 22, 349, 1993.

[35] Grove, E. F., Online bin packing with lookahead, *Proc. SODA*, SIAM, 1995, p. 430.

[36] Baker, B. S., and Coffman, Jr. E. G., A tight asymptotic bound for next-fit-decreasing bin-packing, *SIAM J. Alg. Disc. Meth.*, 2, 147, 1981.

[37] Baker, B. S., A new proof for the first-fit decreasing bin-packing algorithm, *J. Algorithms*, 6, 49, 1985.

[38] Yue, M., A simple proof of the inequality $\text{FFD}(L) \leq \frac{11}{9}\text{OPT}(L) + 1\ \forall L$, for the FFD bin-packing algorithm, *Acta Math. Appl. Sinica*, 7, 321, 1991.

[39] Li, R. H. and Yue, M. Y., The proof of FFD(L) ≤ 11/9OPT(L) + 7/9, *Chinese Sci. Bull.*, 42, 1262, 1997.

[40] Csirik, J., The parametric behavior of the first-fit decreasing bin packing algorithm, *J. Algorithms*, 15, 1, 1993.

[41] Xu, K., The asymptotic worst-case behavior of the FFD heuristics for small items, *J. Algorithms*, 37, 237, 2000.

[42] Friesen, D. K. and Langston, M. A., Analysis of a compound bin-packing algorithm, *SIAM J. Disc. Math.*, 4, 61, 1991.

[43] Garey, M. R. and Johnson, D. S., A 71/60 theorem for bin packing, *J. Complexity*, 1, 65, 1985.

[44] Graham, R. L., Bounds on multiprocessing anomalies and related packing algorithms, *Proc. Spring Joint Computer Conf.*, AFIPS Press, Arlington, VA, 1972, p. 205.

[45] Caprara, A. and Pferschy, U., Worst-case analysis of the subset sum algorithm for bin packing, *Oper. Res. Lett.*, 32, 159, 2004.

[46] Blazewicz, J. and Ecker, K., A linear time algorithm for restricted bin packing and scheduling problems, *Oper. Res. Lett.*, 2, 80, 1983.

[47] Lenstra, H. W., Jr., Integer programming with a fixed number of variables, *Math. Oper. Res.*, 8, 538, 1983.

[48] Gilmore, P. C. and Gomory, R. E., A linear programming approach to the cutting stock problem, *Oper. Res.*, 9, 839, 1961.

[49] Gilmore, P. C. and Gomory, R. E., A linear programming approach to the cutting stock program—Part II, *Oper. Res.*, 11, 863, 1963.

[50] Hochbaum, D. S. and Shmoys, D. B., A packing problem you can almost solve by sitting on your suitcase, *SIAM J. Alg. Disc. Meth.*, 7, 247, 1986.

[51] Leung, J. Y.-T., Bin packing with restricted piece sizes, *Inf. Proc. Lett.*, 31, 145, 1989.

[52] Coffman, E. G., Jr., Garey, M. R., and Johnson, D. S., Bin packing with divisible item sizes, *J. Complexity*, 3, 405, 1987.

[53] Martel, C. U., A linear time bin-packing algorithm, *Oper. Res. Lett.*, 4, 189, 1985.

[54] Békési, J., Galambos, G., and Kellerer, H., A 5/4 linear time bin packing algorithm, *JCSS*, 60, 145, 2000.

[55] Kenyon, C., Best-fit bin-packing with random order, *Proc. SODA*, SIAM, 1996, p. 359.

[56] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, New York, 1979.

[57] Garey, M. R., Graham, R. L., Johnson, D. S., and Yao, A. C., Resource constrained scheduling as generalized bin packing, *J. Combin. Theor. Ser. A*, 21, 257, 1976.

[58] Simchi-Levi, D., New worst-case results for the bin packing problem, *Nav. Res. Log.*, 41, 579, 1994.

[59] Xie, J. and Liu, Z., New worst-case bound of first-fit heuristic for bin packing problem (Unpublished manuscript).

[60] Berghammer, R. and Reuter, F., A linear approximation algorithm for bin packing with absolute approximation factor 3/2, *Sci. Comput. Programming*, 48, 67, 2003.

[61] Zhang, G., X. Cai, X., and Wong, C. K., Linear time-approximation algorithms for bin packing, *Oper. Res. Lett.*, 26, 217, 2000.

[62] Murgolo, F. D., Anomalous behavior in bin packing algorithms, *Disc. Appl. Math.*, 21, 229, 1988.

[63] Fortune, S. and Wyllie, J., Parallelism in random access machines, *Proc. STOC*, ACM Press, New York, 1978, p. 114.

33 Variants of Classical One-Dimensional Bin Packing

Edward G. Coffman, Jr.
Columbia University

János Csirik
University of Szeged

Joseph Y.-T. Leung
New Jersey Institute of Technology

33.1 Introduction

Few problems compete with the bin packing problem in having fascinated so many people for so long a time. Research into the classical bin packing problem dates back over three decades to the early 1970s. In the original version, a list $L = (a_1, a_2, \ldots, a_n)$ of n items, each with a size no larger than 1, is given along with an infinite supply of unit capacity bins. The goal is to pack the list into as few bins as possible so that no bin capacity is exceeded. Because the problem is NP-hard, most research has concentrated on designing fast approximation algorithms with good performance guarantees. The studies have spanned both online and offline algorithms, and have applied both combinatorics and probabilistic analysis.

In parallel with the development of approximation algorithms for the classical problem, several variants have been proposed and studied. In this and the next chapter we survey some of the results for these variants. Because of the extensive work done in this area, we cannot possibly hope to cover every problem. Rather, we choose some of the representative ones, and hope that they will give the reader a flavor of the richness of the problem area.

In Section 33.2, we survey the variant in which the number of items packed is maximized. In this problem the number of bins, m, is fixed and the goal is to pack a maximum number of pieces into the m bins. This problem was first proposed by Coffman et al. [1] in 1978, and studied more recently by Boyar et al. [2].

Section 33.3 surveys the variant that places a bound on the number of items that can be packed in each bin. This problem is identical to the classical bin packing problem, except that, for given $k > 0$, each bin can contain at most k items. The problem was first proposed and studied by Krause et al. [3] in 1975.

In Section 33.4, we survey dynamic bin packing, in which packings change with time; each item has an arrival and departure time, which define the time interval during which an item occupies a bin. This problem was first studied by Coffman et al. [4] in 1983.

Section 33.5 surveys several variants that put constraints on the data. The first problem, first studied by Liu and Sidney [5], is concerned with bin packing under the ordinal data scenario: the item sizes are not known but the order of the weights is known. The second problem, first proposed by Mandal et al. [6], allows the items to be fragmented while packing them into bins of fixed capacity. The last problem, first studied by Jansen and Öhring [7], considers the bin packing problem where certain items cannot be put into the same bin. An undirected graph $G = (V, E)$ is used to describe these relationships, where V is a set of vertices representing the items and E a set of edges such that items connected by an edge $(a_i, a_j) \in E$ have to be packed into different bins.

In the final section, we survey the bin stretching problem, first studied by Azar and Regev [8]. In this problem, the optimal bin capacity is known and the goal is to find a good heuristic, given this information. The supremum of the bin capacity ratio of an algorithm and the optimal value is the stretching factor. The goal is to find a simple algorithm with a small stretching factor.

In the next chapter, we survey bin packing into bins of different sizes. The variable-sized bin packing problem was first studied by Friesen and Langston [9] in 1986, and it has attracted much attention since then. We then survey bin covering problems, which ask us to partition a set of items into the maximum number of subsets such that, in every subset, the total size of the items is never less than some lower bound. The problem was first studied by Assman et al. [10] in 1984.

33.2 Maximizing the Number of Items Packed

Coffman et al. [1] introduced the following bin packing variant to model processor and storage allocation problems. For example, it might be desirable to maximize the number of records stored in multiple, autonomous storage units, or to maximize the number of tasks that can be executed on multiple processors during a fixed time interval. The formal model is as follows: given a set of m unit capacity bins $B_1, B_2, \ldots, B_m$ and a list of $L = (a_1, a_2, \ldots, a_n)$ of items, the goal is to pack a maximum subset of L into the bins such that no bin capacity is exceeded. It is intuitively clear that any algorithm having a reasonable worst-case performance relative to an optimal algorithm must attempt to pack a maximum number of the smaller pieces. So it is quite natural to start by sorting pieces in ascending order of size and then to pack a maximum prefix of the sorted list. It is obvious that for every sublist $L' \subseteq L$ that can be packed into m bins there is a prefix of L having at least as many pieces that can also be packed into the m bins. Coffman et al. always used the sorted lists for their algorithms.

Let n_O denote the optimal number of items packed into m bins and n_A the number of items packed by an algorithm A. Coffman et al. considered the absolute worst-case ratios, giving bounds on n_A/n_O. They first considered the SMALLEST-PIECE-FIRST (SPF) algorithm, which is defined as follows: the list is sorted in ascending order of piece size and each piece is assigned in turn to the bin having the least filled space, with ties broken in favor of the bin having the lowest index. The algorithm halts when it first encounters a piece that is too large to fit in any bin. It is quite easy to see that the asymptotic worst-case performance of SPF is 1/2.

The next algorithm they investigated was the FIRST-FIT-INCREASING algorithm (FFI), that is, the First Fit (FF) rule applied to the sorted list. They proved that, for any list packed into m bins,

$$\frac{n_{FFI}}{n_O} \geq \frac{3}{4}$$

and that, for every m, there exists a list which realizes this bound. A straightforward example can be given as follows:

$$\Bigg(\underbrace{\frac{1}{2} - \varepsilon, \ldots, \frac{1}{2} - \varepsilon}_{m \text{ items}}, \underbrace{\frac{1}{2} + \varepsilon, \ldots, \frac{1}{2} + \varepsilon}_{m \text{ items}}\Bigg)$$

where m is even. The optimal packing will pack all items ($2m$) into m bins, while FFI will pack $3m/2$ items.

They also provided a more precise bound from which parametric results can be derived. Let k_i denote the number of items in bin B_i in the FFI packing. For FFI, it is easy to see that $k_1 \geq k_2 \geq \cdots \geq k_m$. With the help of k_m they proved that

$$n_{FFI} \geq \begin{cases} \frac{mk_m+1}{m(k_m+1)} n_O, & m \leq k_m + 1 \\ \frac{k_m^2+k_m+1}{(k_m+1)^2}(n_O - 1), & m > k_m + 1 \end{cases}$$

Both bounds are attainable.

Coffman and Leung [11] described a better algorithm for this problem, which is much more difficult to analyze. They introduced the ITERATED-FIRST-FIT-DECREASING rule (FFD*) that works as follows. Again, assume that the elements of L are in ascending size order. FFD* first scans L to find the maximum length prefix $L^{(1)} = (a_1, a_2, \ldots, a_t) \subseteq L$ such that $\sum_{i=1}^{t} a_i \leq m$. The algorithm then packs $L^{(1)}$ into as many bins as required (say m'), by scanning right to left and placing the next smaller piece into that bin with the lowest index into which it will fit. The algorithm terminates successfully if $m' \leq m$; otherwise, the algorithm constructs $L^{(2)} \subset L^{(1)}$ by discarding the largest piece in $L^{(1)}$ and then proceeds as above to pack $L^{(2)}$ by the FFD rule. The process is repeated until, for some j, $L^{(j)}$ has been packed into $m' \leq m$ bins. They proved that, for all L and $m \geq 1$,

$$n_{FFD^*}(L, m) \geq \frac{6}{7} n_O(L, m) - \frac{18}{7}$$

and there exist lists for each m such that FFD* will pack 7/8 times the optimum number of pieces. A tight bound has not yet been found.

These are, of course, not online algorithms. It is quite easy to see that without some restriction on the lists to be processed, all online algorithms can perform very poorly; just take lists starting with large items so that, after all of the bins are filled with these items, many very small items remain. Here we need some restrictions so as to have reasonably good online algorithms. With the classical bin packing problem all items have to be packed, so it seems natural to restrict the input sequences to those that can be completely packed by an optimal offline algorithm. Such sequences are called *accommodating sequences* [2].

Boyar et al. [2] considered *greedy* algorithms: an item is rejected only if there is not enough space to pack it (the authors called them fair algorithms). Boyar et al. analyzed the FF algorithm and proved that it has a competitive ratio of at least 5/8 on accommodating sequences. Azar et al. [12] showed that this bound is asymptotically tight, that is, the competitive ratio on accommodating sequences comes arbitrarily close to 5/8 for large enough n. Specifically, for any n, the competitive ratio is bounded by $5/8 + O(1/\sqrt{n})$.

Azar et al. [12] investigated *unrestricted* algorithms that have the power of performing admission control on the pieces, that is, rejecting pieces while there is enough space to pack them. They designed an algorithm for unrestricted bin packing called UNFAIR-FIRST-FIT (UFF) whose competitive ratio is $2/3 \pm \theta(1/n)$. UFF uses two sets of pieces: A (accepted) and R (rejected). Since every piece has the same value, it seems reasonable to reject large pieces. So UFF examines whether the piece is larger than 1/2 and whether the performance ratio would still be at least 2/3 if the piece were rejected. If both conditions are satisfied, the piece is rejected (placed in R); otherwise, it is accepted (placed in A). All accepted pieces will be packed according to the FF rule.

Azar et al. [12] also showed that the competitive ratio of any online algorithm on accommodating sequences is no better than $0.857 + O(1/n)$ even when randomized algorithms are considered. For deterministic fair algorithms, they proved a slightly better bound of $0.809 + O(1/n)$.

Boyar et al. [2] gave additional results for the following special case: they assumed that the item size and bin size were integers. Let k be the bin size here. They investigated special lists where they know that the optimal offline algorithm would use no more than α times the given number of bins, with $\alpha \geq 1$ (for $\alpha = 1$ we get the accommodating sequences). They called these sequences α-*sequences.* Algorithm A is called c-accommodating with respect to α-sequences if $c \leq 1$ and for every α-sequence L,

$$A(L) \leq c \cdot OPT(L) - b$$

where b is a fixed constant. Let $\mathcal{C}_\alpha = \{c \mid A \text{ is } c\text{-accommodating with respect to } \alpha\text{-sequences}\}$. The *accommodating function* $\mathcal{A}$ is defined as $\mathcal{A}(\alpha) = \sup\, \mathcal{C}_\alpha$.

To maximize the number of pieces packed, Boyar et al. proved that if $\alpha \le 5/4$ and $k \ge 3$, then

$$\mathcal{A}(\alpha) \le \frac{2}{2 + (\alpha - 1)(k - 2)}$$

Boyar et al. [13] used the same restrictions—integer-sized items with an integer bin capacity of k. Under these conditions it is quite easy to see that the competitive ratio of FF is $1/k$. They introduced a new algorithm Dlog and compared it with FF from various points of view. The Dlog algorithm works as follows. Let us first suppose that we have m bins. We assume that $m \ge c\lfloor \log_2 k \rfloor$, for some constant c. Divide the bins into $\lceil \log_2 k \rceil$ groups $G_1, G_2, \ldots, G_{\lceil \log_2 k \rceil}$. Let $p = \lfloor \frac{n}{\lceil \log_2 k \rceil} \rfloor$ and let $s = n - p\lceil \log_2 k \rceil$. Groups $G_1, G_2, \ldots, G_s$ consist of $p + 1$ bins and the other groups consist of p bins. Let $S_1 = \{x \mid \frac{k}{2} \le x \le k\}$ and $S_i = \{x \mid \frac{k}{2^i} \le x \le \frac{k}{2^{i-1}}\}$ for $2 \le i \le \lceil \log_2 k \rceil$. When Dlog receives an item o of size $s_o \in S_i$, it decides which G_j group of bins to pack it in by calculating $j = \max\{j \le i \mid \text{there is a bin in } G_j \text{ that has room for } o\}$. If j exists, o is packed into G_j according to the FF rule. Otherwise, the item o is rejected.

The authors proved that for every $\alpha \ge 1$, if $m \ge c\lfloor \log_2 k \rfloor$, the competitive ratio of Dlog on α-sequences is $\Theta(\frac{1}{\log k})$.

They also defined a randomized version of log and proved that, on accommodating sequences, its competitive ratio is $\Theta(\frac{1}{\log k})$. This is quite close to the best possible, as they have already shown that any deterministic or randomized online algorithm for this problem has a competitive ratio of less than $\frac{4}{\lfloor \log_2 k \rfloor}$. Some other results were given where we know the optimum value in advance.

Coffman et al. [14] investigated the previous FFI and ITERATED-FIRST-FIT-DECREASING algorithms for lists with divisibility conditions. They proved that if L is weakly divisible, then FFD* will produce an optimal packing. Moreover, if L is strongly divisible, even FFI gives an optimal packing.

33.2.1 Generalizations of the Objective Function

In the classical bin packing problem the cost associated with a given bin is either zero or one, depending on whether it is empty or not. Anily et al. [15] defined a more general model, where the cost of a bin is a monotone and concave function of the number of items in a bin. Let $f(j)$ define the cost of a bin that contains j items. The properties are defined as

Monotonicity. The cost of a bin does not decrease with the inclusion of additional items, i.e., if $j \le k$, then $f(j) \le f(k)$.

Concavity. The incremental cost (due to the addition of an item) to a set of items is no more than the incremental cost resulting from the addition of an item to a smaller set. Formally, for all $j \ge 1$, $f(j + 1) - f(j) \le f(j) - f(j - 1)$. We assume that $f(0) = 0$ as well.

The authors showed that the classical algorithms NF, FF, BF, FFD, and BFD have neither finite absolute worst-case ratios nor finite asymptotic worst-case ratios for the bin packing problem with general cost structure, where these ratios are defined in a similar way to those for the classical problem—just replace the number of bins by the sum of the cost of all bins used. They also proved that the absolute worst-case bound for the NFI (NEXT-FIT-INCREASING) algorithm is less than or equal to

$$\min\left\{1.75,\, 1.7 + \frac{2}{b_{OPT}(L)},\, 1.691\ldots + \frac{3}{b_{OPT}(L)}\right\}$$

where $b_{OPT}(L)$ denotes the optimal cost for the list L. This is the best known algorithm for this problem.

33.3 Bounds on the Number of Items per Bin

Krause et al. [3] investigated a task scheduling problem on a multiprogramming computer system. Here we suppose that we have a system with $k \ge 2$ processors that share a common memory of fixed capacity. A sequence of tasks with unit processing times have to be executed on these processors. Each task has a certain memory requirement. The goal here is to execute these tasks in the shortest possible time. We can represent the tasks by items and each unit of time by one bin. The memory requirements of the tasks are then the item sizes, and tasks that are performed at the same time correspond to items in the same bin. The size of the memory is the bin capacity. The schedule length will be the number of bins used in the packing. So they actually defined a bin packing problem with the restriction that no more than k items can be placed in one bin. Krause et al. at first investigated a suitable modification of the FF heuristic: use the FF rule with the additional constraint that we can place an item into a bin only if the number of items is not larger than k. Let us call this heuristic kFF. They proved that for each list L,

$$kFF(L) \leq \left(\frac{27}{10} - \frac{24}{10k}\right) OPT(L) + 2$$

and also gave examples (with arbitrarily large OPT(L)) such that

$$kFF(L) = \left(\frac{27}{10} - \left\lceil \frac{37}{10k} \right\rceil\right) OPT(L)$$

From this we have

$$\left(\frac{27}{10} - \left\lceil \frac{37}{10k} \right\rceil\right) \leq R^{\infty}_{kFF} \leq \left(\frac{27}{10} - \frac{24}{10k}\right)$$

where, in particular, $R^{\infty}_{2FF} \leq 1.5$ and $1.4666\ldots \leq R^{\infty}_{3FF} \leq 1.9$.

This is, of course, an online algorithm. They also offered better offline algorithms. The first is called the LARGEST MEMORY FIRST (LMF) algorithm, but it is actually a suitable modification of the First Fit Decreasing rule: first sort the items and then use the FFD rule. If a bin already has k items, it is closed. For this algorithm they proved that

$$R^{\infty}_{LMF} = 2 - \frac{2}{k}$$

if $k \geq 2$.

The second algorithm of Krause et al. is called the ITERATED WORST-FIT DECREASING (IWFD) rule. It starts by sorting the items in ascending order of size and then opens q empty bins, where

$$q = \left\lceil \max\left(\frac{n}{k}, \sum_{j=1}^{n} s(a_j)\right) \right\rceil$$

n being the number of items in our list. It is an obvious lower bound on $OPT(L)$. The IWFD rule uses the Largest Processing Time (LPT) rule to pack the list: put the next item into the bin with the lowest level. It breaks ties in favor of the highest index. The IWFD rule starts with q bins. If LPT does not pack all items into q bins, then this number is increased by 1. All bins are cleared and the procedure starts all over again. This is repeated until the whole list is packed. The number of bins used in the last step is defined as IWFD(L). It can be shown that for special lists the IWFD rule has good properties and for the general case

$$\frac{4}{3} \leq R^{\infty}_{IWFD} \leq 2$$

As the gap is quite large, the exact value remains an interesting open question. The authors put it at 4/3.

Babel et al. [16] returned to this bin packing problem. They presented two heuristics for the online version. Both heuristics better the previous best result by Krause et al. For increasing values of k, the asymptotic worst-case ratio tends to 2 for the first method, and it is 2 for the second algorithm. The chief feature of the first heuristic is that a new item can be packed into a bin only if the bin contains relatively few items or, if it does not, the bin together with this new item is sufficiently filled. The second algorithm defines three different bin types (depending on the bin level and the number of items already packed) and attempts to pack the next item first in the first, then in the second, and finally in the third type of bins. The authors also gave the best possible algorithm for $k = 2$ with an asymptotic worst-case ratio of $\sqrt{2}$. For $k = 3$ they offered a simple heuristic with worst-case ratio of 5/3 and showed that no online algorithm for this case can have worst-case ratio better than 3/2.

33.3.1 Online Bin Coloring

Shachnai and Tamir [17] defined a coloring problem, which is actually a generalization of bounds on the number of items per bin. In their version every item has the same (unit) size and a color. The number of different colors is M. The bins have a capacity B and c compartments. We need to fill the bins with items, subject to capacity constraints (such that items of different colors are placed in different compartments) so that each bin contains at most c different colors. The goal is to minimize the number of bins used. The absolute and asymptotic worst-case ratios are defined in a similar way to the classical problem. This variant is called the *Class-Constrained Bin Packing* (CCBP). Their first result is a tight bound of 2 on the competitive ratio of any online algorithm.

For special lists they obtained better results. Let $S_{c,B}(k, h)$ denote the set of all lists, where $kc < M \leq (k+1)c$ and $h \cdot B < n \leq (h+1) \cdot B$ (n is the number of items in the list). The competitive ratio of an algorithm A restricted to an input set S of lists will be denoted by $r_A(S)$. They showed that for any deterministic algorithm A, $1 < c < B$, $k \geq 0$ and $h \geq k - 1 + \max\{\lceil k/(c-1)\rceil, \lceil (kc+1)/B\rceil\}$,

$$r_A(S_{c,B}(k, h)) \leq 1 + \frac{k + 1 - \left\lceil \frac{kc+1}{B} \right\rceil}{h+1}$$

The authors showed that a variant of the FF rule achieves this bound. A greedy algorithm based on partitioning the items into color sets was shown to be efficient as well.

Krumke et al. [18] introduced a new variant of bin packing called *bin coloring*. Here we have unit size items where each item has a color, and we have to pack them into bins of size B (an integer). They considered an online, bounded-space problem, that is, one where we can use at most m open bins at the same time. A further restriction is that we can close a bin only when it is full, that is, after we have packed B items into it. The goal is to minimize the maximum number of different colors assigned to a bin. To measure the goodness of a heuristic algorithm they used the absolute worst-case ratio. They investigated two heuristic algorithms and gave some general bounds.

- The first method was the GREEDYFIT rule, which works as follows: if upon the arrival of item a_i the color of this item is already contained in one of the currently open bins, put it into this bin. (If the bin becomes full with this item, close it.) Otherwise, put the item into a bin that contains the least number of different colors (which means opening a new bin if currently fewer than m bins are partially filled). The competitive ratio of this method is at most $3m$ but not less than $2m$ (provided the capacity B is sufficiently large).
- The second method was the trivial algorithm ONEBIN, which is actually the Next Fit algorithm: the next item is packed into the (one) open bin. A new bin is opened only if the previous item closed the previous bin by filling it up completely. Quite surprisingly, this algorithm has a better competitive ratio than GREEDYFIT, it being $2m - 1$.

They also proved that no online algorithm can be substantially better than ONEBIN: no deterministic online algorithm can have a competitive ratio better than m. This bound holds for randomized algorithms too. Using resource augmentation, that is, allowing the online algorithm to use a fixed number of $m' \geq m$ open bins, the lower bound will still be $\Omega(m)$.

33.4 Dynamic Bin Packing

Coffman et al. [4] defined an interesting generalization of the classical problem. They added arrival and departure times to each item. These define the various time intervals during which items occupy bins. Compared with the classical "archival" model, this model is more realistic in certain applications, such as those relating to computer storage allocation.

The formal model is defined as follows. We are given a list $L = (p_1, p_2, \ldots, p_n)$. Each item p_i in L corresponds to a triple (a_i, d_i, s_i), where a_i is the arrival time for p_i, d_i its departure time, and s_i its size. The item p_i resides in the packing for the time interval $[a_i, d_i)$, and we assume that $d_i - a_i > 0$ for all i. We also assume that the items in L are ordered in such a way that $a_1 \leq a_2 \leq \cdots \leq a_n$.

Coffman et al. analyzed the FF method for dynamic bin packing. With this version, FF maintains two lists of bins: a list of currently empty bins and a list of currently occupied bins, with the latter ordered such that the times of the most recent transitions from empty to nonempty bins are nondecreasing. We use $B_1, B_2, \ldots$ to denote the bins in the occupied bins, in this order.

For each i, as in the classical version, the FF rule attempts to pack p_i into the first occupied bin (the one with the lowest index) that has sufficient available space for it. If no such occupied bin exists, p_i is put into an empty bin and this bin is appended to the list of occupied bins. The departure of p_i at time d_i simply causes an increase by s_i in the available space in the bin in which it was packed and, if the bin becomes empty at that time, it is moved to the list of empty bins.

Let $FF(L, t)$ denote the number of occupied bins in the FF packing of L at time t. The performance of FF applied to the list L is defined by

$$FF(L) = \max_{0 \le t \le a_n} FF(L, t)$$

This is the maximum number of occupied bins ever required by FF for processing L. This definition can be applied to an arbitrary packing algorithm A simply by replacing FF with A. We will apply this measure to two different ways of optimal packing depending on how the items in L can be packed. The first method, $OPT_R(L)$, is the maximum number of bins ever required in dynamically packing L when the current set of items is *repacked* into the minimum number of bins each time a new item arrives. The second, $OPT_{NR}(L)$, is the maximum number of bins ever required in dynamically packing L when *no rearrangement* of items is allowed, and otherwise packed optimally so as to achieve the least possible value of this maximum over all such packings of L, with L assumed to be fixed and known in advance. The performance ratio of an arbitrary packing algorithm A is now defined in the standard way. Let

$$W_{R,n}\left(A, \frac{1}{k}\right) = \sup_{|L|=n} \frac{A(L)}{OPT_R(L)}$$

where all item sizes are less than or equal to $1/k$. We then define

$$W_R^{\infty}\left(A, \frac{1}{k}\right) = \limsup_{n \to \infty} W_{R,n}\left(A, \frac{1}{k}\right)$$

Similar definitions apply to $W_{NR}^{\infty}(A)$. The authors gave the following bounds:

$$W_R^{\infty}\left(FF, \frac{1}{k}\right) \le \frac{k+1}{k} + \frac{1}{k-1} \log \frac{k^2}{k^2-k+1}, \quad k \ge 2$$

$$W_R^{\infty}(FF, 1) \le \frac{5}{2} + \frac{3}{2} \log \frac{\sqrt{13}-1}{2} \approx 2.897$$

They also gave a slightly improved FF version for which W_R^{∞} is bounded by 2.788.... In addition, they proved that

$$W_R^{\infty}\left(FF, \frac{1}{k}\right) \ge \frac{k+1}{k} + \frac{1}{k^2}, \quad k \ge 2$$

$$W_R^{\infty}(FF, 1) \ge \frac{11}{4} = 2.75$$

They derived a lower bound that holds for an *arbitrary* online algorithm A :

$$W_R^{\infty}\left(A, \frac{1}{k}\right) \ge W_{NR}^{\infty}\left(A, \frac{1}{k}\right) \ge \frac{k+1}{k} + \frac{1}{k(k+1)}, \quad k \ge 2$$

$$W_R^{\infty}(A, 1) \ge \frac{5}{2}$$

$$W_{NR}^{\infty}(A, 1) \ge \frac{43}{18} \sim 2.388\ldots.$$

We should note that the upper and lower bounds are quite close to each other. We have the largest gap for $W_R^\infty(FF, 1)$ for which

$$2.75 \le W_R^\infty(FF, 1) \le 2.897$$

33.5 Constraints on Data

33.5.1 Bin Packing Using Semi-Ordinal Data

Liu and Sidney [5] investigated the bin packing problem under the ordinal data scenario: the item sizes are not known, but the order of the weights is known, that is, $a_1 \ge a_2 \ge \cdots \ge a_n$. Specific sizes may be obtained, and the trade-off between the amount of information obtained versus the solution quality (performance ratio) is investigated.

If the only information available about L is the ordinal assumption, then we can place only one item per bin, since all items may be larger than 1/2. Even in this case the optimal solution may be 1 if $\sum a_i \le 1$. So we have a worst-case ratio of n. Suppose we could purchase perfect knowledge of k sizes, where we can specify the ranks (ordinal positions) of the desired sizes. Which ordinal positions should we choose to guarantee a good feasible solution that utilizes the knowledge of these sizes? Liu and Sidney proved that the worst-case ratio of an integer ρ can be achieved by at most

$$\lfloor \ln[n(\rho - 1) + 1]/\ln\,\rho \rfloor$$

exact observations, and that this worst-case performance cannot be achieved with fewer than

$$\left\lfloor \frac{\ln \frac{n(\rho-1)}{\rho^2-\rho+1}}{\ln\,\rho} \right\rfloor + 1$$

exact observations. They also gave a method for choosing a good set of indices.

33.5.2 Fragmentable Bin Packing

Mandal et al. [6] defined the following variant called *Fragmentable Object Bin Packing*: it is permissible to fragment the items while packing them into bins of fixed capacity. Fragmentation has an additional cost associated with it, leading to the consumption of additional bin capacity. They proved that this problem is NP-hard too. Unfortunately no heuristic was analyzed from the worst-case point of view.

Menakerman and Rom [19] tackled this problem further and defined two variants. The first variant is called BIN PACKING WITH SIZE-INCREASING FRAGMENTATION (BP-SIF). In this variant, when packing a fragment of an item, one unit of overhead is added to the size of every fragment. They supposed that the item sizes were integer-valued and that each bin size had an integer value U. An algorithm is said to prevent unnecessary fragmentation if it follows the following two rules:

- *No unnecessary fragmentation.* An item (or fragment of an item) is fragmented only if it is to be packed into a bin that cannot contain it. In this case the item is fragmented into two fragments. The first fragment must fill one of the bins, and the second fragment must be packed according to the packing rule of the algorithm.
- *No unnecessary bins.* An item is packed into a new bin only if it cannot fit in any of the open bins used by the algorithm (greedy).

They proved that for any algorithm that prevents unnecessary fragmentation the absolute worst-case bound is $\le \frac{U}{U-2}$. They also showed that the following variant of the NEXT FIT rule, called NF_f, achieves this bound. When an item does not fit in the open bin, it is fragmented into two parts. The first part or piece fills the open bin and the bin is closed. The second part or piece is packed into a new bin which becomes the open bin. The Next Fit Decreasing and the Next Fit Increasing rules have basically the same bound, for large enough U. Neither the First Fit Decreasing rule nor the following iterative variant of the

First Fit Decreasing rule improves the bound: we start with $m = \lceil s(L)/U \rceil$ bins and try to pack the items by the FFD rule. If every item has been packed then we stop. If some remain then m is increased by 1 and we start packing the whole list again. This is repeated until every item has been packed.

The second variant is BIN PACKING WITH SIZE-PRESERVING FRAGMENTATION (BP-SPF). In this version each item has a size and a cost. The items must be packed into m bins, where $s(L) \leq mU$. It is possible to fragment any item, in which case one unit is added to its cost while keeping the size fixed. The goal is to minimize the total cost. As every item must be packed and for the chosen m this is possible because the fragmentation does not increase the size we have a fixed cost for the whole list. The minimization of the cost is equivalent to the minimization of the number of fragmentations. The problem remains NP-hard. They proved that for any algorithm A that has no unnecessary fragmentations, the extra cost is bounded above by $m - 1$, and that NF_f achieves this bound. The appropriate variant of the FFD rule betters this bound only when U is small. If the bin size is unbounded, the worst-case behavior of the FFD rule is the maximum possible value.

33.5.3 Packing Nodes of Graphs

Jansen and Öhring [7] once looked at a constrained bin packing problem: in this case there is an undirected graph (*the conflict graph*) $G = (J, E)$ on the elements. The adjacent items $(a_i, a_j) \in E$ have to be packed into different bins, and the goal is to minimize the number of bins. Clearly, if E is an empty set, the problem is equivalent to the classical bin packing problem. In contrast, if $s(L) \leq 1$, then the problem is to compute the chromatic number $\chi(G)$ of the conflict graph. Both special cases are NP-complete. The authors used the absolute worst-case ratios and then proved that simply using the appropriate modifications of the classical algorithms NF, FF, and FFD will not yield a constant competitive ratio. (The appropriate modification simply means that we use the heuristic, but if the item to be packed has an adjacent item in the bin being packed, then we do not pack it into this bin.) Their other algorithms were based on the composition of two algorithms—a coloring algorithm and a bin packing heuristic.

The first algorithm of this type uses an optimal coloring, that is, it finds a minimum partition of L into independent sets $U_1, U_2, \ldots, U_{\chi(G)}$, and then applies one of the NF, FF, and FFD bin packing heuristics to each independent set. They proved that in this case NF has a competitive ratio of 3, FF of 2.7, and for the FFD the bound lies between 2.691 and 2.7.

The second algorithm of this type is based on a precoloring method. The main step is to compute a minimum coloring of the conflict graph where the large items are separated and colored differently. On the basis of this method, they obtained an approximation algorithm with competitive ratio of 5/2 for graphs such as interval graphs, split graphs, cographs, and other graphs. More involved coloring methods are used to obtain algorithms with worst-case bounds of 7/3, 11/5, and 15/7, respectively. The last of these methods is a general separation method that works for cographs and partial K-trees. Applying this separation method they obtained an approximation algorithm with worst-case ratio of $2 + \varepsilon$ for these two classes of graphs. This result implies an approximation with factor $2 + \varepsilon$ for any class of graphs with a constant upper bound on the treewidth (e.g., outerplanar graphs and series-parallel graphs).

Katona [20] investigated the edge disjoint polyp packing problem. A graph is called a *p-polyp* if it consists of p simple paths of the same length and one vertex of these paths is a common vertex. The polyp packing problem is a generalization of the classical bin packing problem: how does one pack a set of paths with different lengths into a set of disjoint polyps edges? Edge disjoint packing means that the embedded path may contain a vertex more than once, but each edge only once. Katona proved that this problem is NP-complete and that the appropriate modification of the FF rule has an asymptotic worst-case bound lying between 1.6904 and 1.7.

Codenetti et al. [21] have investigated the following variant: the items to be packed are structured as the leaves of a tree. They called this problem a hierarchically *structured bin packing problem* (SBPP). The goal is to pack the items while preserving some locality properties, that is, leaves whose lowest common ancestor has a low height should be packed into the same bin. This problem comes up in the area of document organization and retrieval where one is confronted with searching issues on very large structured, tree-like ontologies. The formal definition is given as follows. Given a tree $T = (V, E)$ and a vertex $v \in T$, $S_T(v)$ will denote the subtree rooted at v, $L_T(v)$ and $N_T(v)$ will denote the set of leaves and internal nodes of $S_T(v)$, respectively. Let $\mathcal{P}$ be a partition of $L_T(v)$. The *node dispersal number* for the node v of T with a given $\mathcal{P}$ is

$$\rho(v, \mathcal{P}) = | \{A \in \mathcal{P} \mid L_T(v) \cap A \neq \emptyset\} |$$

In other words, given a vertex v, $\rho(v, \mathcal{P})$ counts how many sets of $\mathcal{P}$ intersect $L_T(v)$. Now the SBPP is defined as follows. Given a tree T, a positive size function $s : L_T \rightarrow N^+$ and a positive integer (bin) capacity $c \leq \max_{v \in T} s(v)$, find a partition $\mathcal{P}$ of L_T such that

- for every set $A \in \mathcal{P}$ the total size of the leaves in A is bounded by c;
- the total node dispersal number $\rho(\mathcal{P}) = \sum_{v \in N_T} \rho(v, \mathcal{P})$ is minimum.

Informally, we start with a partition $\mathcal{P}$ of the leaves such that, in each subset, the total weight of the leaves is bounded by c (the bin capacity). Then for each internal node of the tree we define the node dispersal number, which is the number of subsets in $\mathcal{P}$ for which the node has a successor in the subset. The total node dispersal number is the sum of the node dispersal numbers for each internal node. In the SBPP we have to find a partition of leaves with the minimal total node dispersal number. The authors presented a heuristic with an asymptotic worst-case ratio of 2. For the special case where every item size is equal to 1 and the bin capacity is k, they found a heuristic with a worst-case bound of 3/2.

33.6 Changing the Bin Size

Azar and Regev [8] investigated the online version of a special case of this problem and called it bin stretching. They supposed that the optimal bin capacity is known and that the goal is to find a good heuristic, given this information. The supremum of the bin capacity ratio of an algorithm and the optimal value is called the stretching factor. They first proved that the stretching factor of any deterministic online algorithm is at least 4/3 for any number $m \geq 2$ of bins. They also gave two algorithms with a stretching factor of 5/3. To describe these algorithms we first need some additional notation. Let $c_j(B_i)$ denote the level of bin i after a_j was packed. Both algorithms use a parameter $\alpha > 0$ to classify the bins according to their levels. An appropriate choice of α will lead to an algorithm with a stretching factor of $1 + \alpha$. We will call a bin *short* if its level is at most α. Otherwise, it is *tall*. When item a_j arrives, $1 \leq j \leq n$, we define the following three disjoint sets of bins:

$$S_1^\alpha(j) = \left\{i \in M \mid c_{j-1}(B_i) + s(a_j) \leq \alpha\right\}$$
$$S_2^\alpha(j) = \left\{i \in M \mid c_{j-1}(B_i) \leq \alpha, \alpha < c_{j-1}(B_i) + s(a_j) \leq 1 + \alpha\right\}$$
$$S_3^\alpha(j) = \left\{i \in M \mid c_{j-1}(B_i) > \alpha, c_{j-1}(B_i) + s(a_j) \leq 1 + \alpha\right\}$$

The set S_1 consists of bins that are short and remain short if the current item is placed into them. The set S_2 contains bins that are short but become tall if the next item is packed. The last set S_3 consists of bins that are tall but remain below $1 + \alpha$ if the next item is packed. Note that there may be bins which are not in any of these three sets. Now the two algorithms are defined as follows:

$ALG1_\alpha$: when the next item a_j arrives

- Put the item into any bin belonging to the set S_3 or S_1, but not in an empty bin belonging to S_1 if there is a nonempty bin belonging to S_1.
- If $S_1 = S_3 = \emptyset$ then put the item into the bin belonging to S_2 with the least level.
- If $S_1 = S_2 = S_3 = \emptyset$ then report failure.

$ALG2_\alpha$: when the next item a_j arrives

- Put the item into any bin belonging to S_1.
- If $S_1 = \emptyset$ then put the item into any bin belonging to S_3.

- If $S_1 = S_3 = \emptyset$ then put the item into the bin with the least level.
- If $S_1 = S_2 = S_3 = \emptyset$ then report failure.

Note that both algorithms actually define a family of algorithms. Azar and Regev proved that both algorithms have a stretching factor of $5/3$ for the best possible choice of α (which is equal to $2/3$). They also presented an improved algorithm using five sets of bins. This has a stretching factor of $13/8$.

Kellerer et al. [22] showed that for two bins the List Scheduling algorithm has a stretching factor of 2.

Epstein [23] gave a tight analysis of bin stretching for two bins, where the items arrive sorted in nonincreasing size. She showed that the tight competitive ratio of this problem is 10/9 and gave a quite complicated algorithm that achieves this bound.

Speranza and Tuza [24] investigated a different problem. Suppose we have m bins of unit capacity and that the bin capacity can be exceeded. If the sum of the sizes of the items in the bin is greater than 1, then the cost of the bin will be this sum. Otherwise, the cost is equal to the capacity 1. The goal is to minimize the total cost of the bins. They investigated the online version of this problem. First they proved that every online algorithm has an absolute worst-case ratio of $7/6$. This can be demonstrated with two bins: simply make the first two elements equal to $1/3$, and for the sum of the next two items, let $a + b = 4/3$. Now, if both $1/3$ items are packed into the same bin, let $a = b = 2/3$. If they are packed into different bins, let $a = 1$ and $b = 1/3$. They proved as well that the List Scheduling algorithm has a tight bound of 5/4.

The above authors also presented an improved algorithm H_x. This depends on a parameter $x, 0 < x < 1$. The algorithm assigns an item to the bin with the highest level under the condition that after the assignment of the item to the bin the level of the bin does not exceed $1 + x$. If the item causes excess greater than x in each open bin, it is then assigned to an empty bin—if there is any—or to the bin with the lowest level. In case of ties, the bin with the lowest index is used. They proved that

$$\frac{H_x(L)}{\text{OPT}(L)} \leq \max\left(1 + \frac{x}{1+x}, \frac{5}{4} - \frac{x^2}{4}\right)$$

The right-hand side has a minimum at $x \approx 0.2956$, the minimum being ≈ 1.228. It is not known whether this bound is tight.

An $13/12$ algorithm is given by Dell'Olmo et al. [25].

Yang and Leung [26] introduced the *Ordered Open-end Bin Packing Problem* (OOBP). Here a bin can be filled to a level exceeding 1 so long as there is a designated last piece in the bin such that the removal of this piece brings the bin's level back to below 1. In this problem items of size 1 play a special role, and will be called 1-pieces. It is quite clear that a good algorithm for this problem will fill the majority of bins to levels no less than 1, while for any algorithm, no bin can be filled to a level more than 2. Hence, any good algorithm will have a worst-case ratio of no more than 2. The authors showed that for any online algorithm A, $R_A^{\infty} \geq 1.630297$ with the 1-pieces, and $R_A^{\infty} \geq 1.415715$ without the 1-pieces.

They then introduced two algorithms. The first is called MIXED FIT (MXF) and is defined as follows: we divide the items into four types according to their size. Type-1 items will have a size $0 < s(a_i) < 1/3$, type-2 items have size $1/3 \leq s(a_i) < 1/2$, type-3 items have size $1/2 \leq s(a_i) < 1$, and type-4 items have size 1. Accordingly, we define a type-1 open bin as an open bin containing a number of type-1 items, a type-2 open bin as a bin containing one or two type-2 items, and a type-3 open bin as a bin containing one type-3 item. We pack the items in order of their arrival. When the current item is of type-1, we pack it into the type-1 open bins using the FF rule without closing any bin. When the current item is of type-2, we pack it similarly into type-2 bins. Suppose the current item is of type-3 or type-4. We pack it into the first type-1 or type-2 open bin whose level is at least $2/3$ and close this bin. If there is no such bin, we pack the item into an open type-3 bin and then close it. If there is no such bin, we pack the item into a new bin and close it if the item is a type-4 item. The authors showed that with no 1-piece present,

$$R_{MXF}^{\infty} \leq 35/18 \approx 1.9444$$

and they also showed that regardless of the presence of the 1-pieces,

$$R_{MXF}^{\infty} \geq 25/13 \approx 1.9231$$

Their second algorithm is the GREEDY LOOK-AHEAD NEXT FIT (GLANF), which is an offline algorithm. With this algorithm there is one open bin at any given moment. GLANF keeps on filling the current bin with items in their original order unless the first piece is a 1-piece or the addition of the next piece will bring the bin's level to be at least 1. For the latter situation GLANF makes some greedy effort in filling the current open bin to the highest possible level. They proved that, without the 1-pieces,

$$\frac{27}{20} \le R^{\infty}_{GLANF} \le \frac{3}{2}$$

and with the 1-pieces,

$$R^{\infty}_{GLANF} \ge \frac{3}{2}$$

References

[1] Coffman, E. G., Jr., Leung, J. Y.-T., and Ting, D. W., Bin packing: maximizing the number of pieces packed, *Acta Informatica*, 9, 263, 1978.

[2] Boyar, J., Larsen, K. S., and Nielsen, M. N., The accommodation function: a generalization of the competitive ratio, *SIAM J. Comput.*, 31, 233, 2001.

[3] Krause, K. L., Shen, Y. Y., and Schwetman, H. D., Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems, *JACM*, 22, 522, 1975.

[4] Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S., Dynamic bin packing, *SIAM J. Comput.*, 12, 227, 1983.

[5] Liu, W.-P. and Sidney, J. B., Bin packing using semi-ordinal data, *Oper. Res. Lett.*, 19, 101, 1996.

[6] Mandal, C. A., Chakrabarti, P. P., and Ghose, S., Complexity of fragmentable object bin packing and an application, *CANDM: Int. J.: Comput. Math. Appl.*, 35, 91, 1998.

[7] Jansen, K. and Öhring, S., Approximation algorithms for time constrained scheduling, *Inf. Control*, 132, 85, 1997.

[8] Azar, Y. and Regev, O., On-line bin-stretching, *Theor. Comput. Sci.*, 268, 17, 2001.

[9] Friesen, D. K. and Langston, M. A., Variable sized bin packing, *SIAM J. Comput.*, 15, 222, 1986.

[10] Assman, S. F., Johnson, D. S., Kleitman, D. J., and Leung, J. Y.-T., On a dual version of the one-dimensional bin packing problem, *J. Algorithms*, 5, 502, 1984.

[11] Coffman, E. G., Jr., and Leung, J. Y.-T., Combinatorial analysis of an efficient algorithm for processor and storage allocation, *SIAM J. Comput.*, 8, 202, 1979.

[12] Azar, Y., Boyar, J., Favrholdt, L. M., Larsen, K. S., Nielsen, M. N., and Epstein, L., Fair versus unrestricted bin packing, *Algorithmica*, 34, 181, 2002.

[13] Boyar, J., Favrholdt, L. M., Larsen, K. S., and Nielsen, M. N., The competitive ratio for on-line dual bin packing with restricted input sequences, *Nordic J. Comput.*, 8, 463, 2001.

[14] Coffman, E. G., Jr., Garey, M. R., and Johnson, D. S., Bin packing with divisible item sizes, *J. Complexity*, 3, 405, 1987.

[15] Anily, S., Bramel, J., and Simchi-Levi, D., Worst-case analysis of heuristics for the bin packing problem with general cost structures, *Oper. Res.*, 42, 287, 1994.

[16] Babel, L., Chen, B., Kellerer, H., and Kotov, V., On-line algorithms for cardinality constrained bin packing problems, in *Proc. of ISAAC*, Eades, P. and Takaoka, T., Eds., Lecture Notes in Computer Science, Vol. 2223, Springer, Berlin, 2001, p. 695.

[17] Shachnai, H. and Tamir, T., Tight bounds for online class-constrained packing, in *Proc. of LATIN*, Rajsbaum, S., Ed., Lecture Notes in Computer Science, Vol. 2286, Springer, Berlin, 2002, p. 569.

[18] Krumke, S. O., de Paepe, W., Rambau, J., and Stougie, L., Online bin-coloring, in *Proc. of ESA*, Meyer auf der Heide, F., Ed., Lecture Notes in Computer Science, Vol. 2161, Springer, Berlin, 2001, p. 74.

[19] Menakerman, N. and Rom, R., Bin packing with item fragmentation, in *Proc. of WADS*, Dehne, F., Sack, J.-R., and Tamassia, R., Eds., Lecture Notes in Computer Science, Vol. 2125, Springer, Berlin, 2001, p. 313.

[20] Katona, Gy. Y., Edge disjoint polyp packing, *Discrete Appl. Math.*, 78, 133, 1997.

[21] Codenetti, B., De Marco, G., Leoncini, M., Montangero, M., and Santini, M., Approximation algorithms for a hierarchically structured bin packing problem, *Inf. Process. Lett.*, 89, 215, 2004.

[22] Kellerer, H., Kotov, V., Speranza, M. G., and Tuza, Zs., Semi on-line algorithms for the partition problem, *Oper. Res. Lett.*, 21, 235, 1997.

[23] Epstein, L., Bin stretching revisited, *Acta Informatica*, 39, 97, 2003.

[24] Speranza, M. G. and Tuza, Zs., On-line approximation algorithms for scheduling tasks on identical machines with extendable working time, *Ann. Oper. Res.*, 86, 491, 1999.

[25] Dell'Olmo, P., Kellerer, H., Speranza, M. G., and Tuza, Zs., A 13/12 approximation algorithm for bin packing with extendable bins, *Inf. Process. Lett.*, 65, 229, 1998.

[26] Yang, J. and Leung, J. Y.-T., The ordered open-end bin packing problem, *Oper. Res.*, 51, 759, 2003.

34

Variable-Sized Bin Packing and Bin Covering

Edward G. Coffman, Jr.
Columbia University

János Csirik
University of Szeged

Joseph Y.-T. Leung
New Jersey Institute of Technology

34.1 Introduction

In this chapter we continue our survey with a focus on two variants of the one-dimensional bin packing problem: the variable-sized bin packing problem and the bin covering problem. In Section 34.2, we survey algorithms for packing into bins of different sizes, a problem first studied by Friesen and Langston [1] in 1986. In Section 34.3, we survey the bin covering problem, which asks for a partition of a given set of items into a maximum number of subsets such that, in every subset, the total item size is always at least some lower bound. This problem was first studied by Assmann et al. [2] in 1984. Concluding remarks are given in Section 34.4.

34.2 Variable-Sized Bin Packing

In the classical bin packing problem the measure of a packing is the number of bins used. The wasted space in the packing is an equivalent measure; a packing that minimizes one of these measures minimizes the other. In what follows, we focus on the wasted-space measure.

Interesting results are available if we relax the condition of having bin sizes fixed in advance. In a problem posed by Friesen and Langston [3], the bin size, which must be the same for all bins, is part of the solution. What is a bin size α, and a packing into bins of this size, which minimizes the wasted space? If there were no bound on the maximum bin size, the problem would be trivial: use only one bin with a size equal to the total item size. But a maximum bin size is a constraint of the problem, and we normalize to one for convenience. The goal is to choose a number $\alpha \leq 1$, and produce a packing of a list $L = (a_1, a_2, \ldots, a_n)$ into N bins of size α such that

$$N\alpha - \sum_{i=1}^{n} s(a_i)$$

is as small as possible. The analysis compares the worst-case wasted space of approximation algorithms to OPT_{α_0}, the minimal space wasted using bin size $\alpha_0 \leq 1$. Friesen and Langston proved that if a bin packing algorithm A can guarantee a worst-case ratio R for the classical bin packing problem, then an iterated version of A can be used to achieve a bound as close to R as we would like. More precisely, if a bound of $R+\epsilon$ is sought, we can generate a sequence of sizes $\alpha_1, \alpha_2, \ldots, \alpha_k$ with $\alpha_{i+1} = \alpha_i R/(R+\epsilon)$ and prove that, if $\alpha_{i+1} < \alpha_0 \le \alpha_i$, then running algorithm A on bins of size α_i will ensure a wasted space $A_{\alpha_i}(L) < (R+\epsilon)OPT_{\alpha_0}(L)$.

The next step taken by Friesen and Langston [1] allowed more than one bin size for packing. They defined the variable-sized bin packing problem on a finite collection of k available bin sizes, $s_1 > s_2 > \cdots > s_k$, with an inexhaustible supply of each size. We adopt the normalization whereby the items $\{a_i\}$ and bins are such that the largest bin has size $s_1 = 1$ and $s(a_i) \le 1$ for all $1 \le i \le n$. The goal is to pack the items into bins so that the sum of the sizes of the bins used is minimum.

The variable-sized bin packing problem is NP-hard [1], so as usual, efficient algorithms that ensure near-optimal packings comprise the design goal. Let $s(A, L)$ denote the total size of the bins used by algorithm A to pack the items of L. Then

$$R_A^{\infty} = \limsup_{k\to\infty} \left\{ \max \frac{s(A, L)}{s(OPT, L)} : s(OPT, L) \ge k \right\}$$

is the asymptotic worst-case ratio of algorithm A.

Friesen and Langston presented three approximation algorithms, with asymptotic worst-case ratios of 2, 3/2, and 4/3, respectively. The first algorithm is a simple adaptation of the NEXT FIT (NF) rule. They called it NEXT FIT USING LARGEST BINS ONLY (NFL), that is, NF packing only into bins of size 1. The proof of $NFL(L) < 2 \cdot OPT(L) + 1$ for any list L is quite standard. Letting ϵ denote an arbitrarily small positive real value, any instance consisting of items of size $1/2+\epsilon$ and bins of sizes 1 and $1/2+\epsilon$ shows that 2 is a matching asymptotic lower bound.

The second algorithm is a variant of the FIRST FIT DECREASING (FFD) rule, called FIRST FIT DECREASING USING LARGEST BINS, THEN REPACK INTO SMALLEST POSSIBLE BINS (FFDLR); and it does what its name says: In the first round the elements are sorted in decreasing order and the items are packed into bins of size 1 by the FFD rule. In the second round it attempts to repack all used bins into the smallest possible bins, where only the full content of a bin can be repacked. This algorithm has an asymptotic worst-case bound of 3/2.

The third algorithm FFDLS (FIRST FIT DECREASING USING LARGEST BINS, BUT SHIFTING AS NECESSARY) uses two rounds as well but tries to repack the contents of certain bins even in the first round. It is a modification of the FFDLR rule in the sense that it uses the same method in both rounds, but in the first round when a_i is packed into bin B_j it checks whether B_j contains an item of size greater than 1/3. If so it repacks, where possible, all the items in B_j into the smallest empty bin $B_{j'}$ that will hold them, and for which $c(B_{j'}) \ge (3/4)c(B_j)$ holds. The second round of the FFDLS rule is the same as that for FFDLR. This algorithm has an asymptotic worst-case bound of 4/3.

Note that only the first of the three algorithms is online. Kinnersley and Langston [4] gave better online algorithms, as follows. First, they proved that two variants of FIRST FIT (FF) (FFL—FIRST FIT, USING LARGEST POSSIBLE BINS and FFS—FIRST FIT, USING SMALLEST POSSIBLE BINS) both have a worst-case bound of two. They observed that the FFL rule fails in its packing of "large" items (those with a size exceeding 1/2) and FFS fails in its packing of "small" items (those with size at most 1/2). So they focused on a hybrid approach that we denote by FFf. Let f denote a user-specified factor in the range $[1/2, 1]$. Suppose FFf has to start a new bin when it is packing a piece a_i. If a_i is a small piece, FFf starts a new bin of size 1. If a_i is a large piece, then it selects the smallest bin size in the range $[s(a_i), s(a_i)/f]$ if such a size exists, otherwise it will use bin size 1. Kinnersley and Langston proved that

$$FFf(L) \le \left(1.5 + \frac{f}{2}\right) OPT(L) + 2$$

for any list L. We should remark here that for the smallest possible f this gives an asymptotic upper bound of 1.75.

They also proved that for the parametric case, FFL gives the same performance as FF for the classical problem, if every item is at most 1/2. Later, Zhang [5] showed that the best possible FFf algorithm (with $f = 1/2$) has a tight bound of 17/10, the bound for FF in the classical bin packing problem.

Csirik [6] introduced a Harmonic type heuristic, VARIABLE HARMONIC M (VH_M) that is based on a harmonic subdivision of each bin size. Let $M > 1$ be a positive integer and let $M_j = \lceil M \cdot s(B_j) \rceil$ ($j = 1, 2, \ldots, k$). The algorithm is defined only for those M where $M_k \geq 2$. Let us divide the intervals $(0, s_j]$ ($j = 1, 2, \ldots, k$) into M_j parts according to the following harmonic partitioning:

$$I_{j,l} = \left(\frac{s_j}{l+1}, \frac{s_j}{l} \right], \quad j = 1, 2, \ldots, k; \; l = 1, 2, \ldots, M_j - 1$$

and

$$I_{j,*} = \left(0, \frac{s_j}{M_j} \right]$$

For each bin size s_j we define a weighting function as follows:

$$W_j(a_i) = \begin{cases} \frac{M_j}{M_j - 1} s(a_i) & \text{if } s(a_i) \in I_{j,*}, \\ \frac{s_j}{l} & \text{if } s(a_i) \in I_{j,l}, \; l = 1, 2, \ldots, M_j - 1, \\ \infty & \text{if } s(a_i) > s_j. \end{cases}$$

Let

$$W(a_i) = \min_{j=1,2,\ldots,k} W_j(a_i)$$

be the weight of a_i.

Now we can define the algorithm VH_M as follows.

Step 1. We assign to each item of the list a bin size; an item assigned to a bin of size s_j is called a type j item.

(a) $s_1 = 1$ is assigned to each element a_i with

$$W(a_i) = (M_1/(M_1 - 1)) \cdot s(a_i)$$

These items are called small items, and all others big items.

(b) A big item a_i will be a type-j element if j is the smallest integer such that $W(a_i) = W_j(a_i)$. A big element will be called an $I_{j,l}$ element if it is a type-j element and belongs to $I_{j,l}$.

Step 2. VH_M performs a harmonic fit type packing:

(a) Each big type-j element is packed by harmonic fit into bins of size s_j as follows. We classify these bins into $M_j - 1$ categories. Each category is designated to pack the same type of items. A bin of size s_j designated to pack $I_{j,l}$ items is called an $I_{j,l}$ bin. Clearly, each $I_{j,l}$ bin has room for exactly l items. We use a NF packing in all $I_{j,l}$ ($j = 1, 2, \ldots, k$; $l = 1, 2, \ldots, M_j - 1$) bins, that is, after packing l items into an $I_{j,l}$ bin, we close this bin and open a new $I_{j,l}$ bin.

(b) All small items are packed in bins of size 1 by NF.

Csirik proved that, using the sequence t_i defined in the harmonic algorithm, the following holds: If $t_{i(k)-1} < M_k \leq t_{i(k)}$, then

$$R_{VH_M} \leq \sum_{l=1}^{i(k)-1} \frac{1}{t_l - 1} + \frac{M_k}{(M_k - 1) t_{i(k)}}$$

He also considered the question of what performance can be achieved if we are free to choose the bin sizes. He proved that if we have just two bin sizes and the smaller bin size is optimally selected (this is 0.7) then the asymptotic worst-case ratio is 1.4.

Seiden [7] showed that variable harmonic is an optimal bounded-space algorithm. Seiden et al. [8] generalized the refined harmonic idea of the classical bin packing problem to the variable-sized problem with two bin sizes. Making use of refined harmonic they bettered the upper bound from 1.69103 to 1.63597. Seiden [9] once demonstrated that if we are allowed to choose the bin sizes then, with two bin sizes, the optimal performance ratio lies in the range [1.37530, 1.37532].

Epstein et al. [10] investigated in more detail the case of two different bin sizes. They presented two different algorithms for this case, both being combinations of harmonic and refined harmonic. They proved that the best of these two algorithms and the variable harmonic yields an upper bound of $373/227 < 1.63597$. They also gave the first lower bound for this problem, showing that for two bin sizes any online algorithm has an asymptotic performance ratio of at least 1.33561.

Burkard and Zhang [11] generalized the bounded space algorithm of Csirik and Johnson [12] so that it applied to the variable bin size problem. In the latter case we have to define the closing rule slightly differently because we may choose in between bin sizes too. Hence their closing rules are:

(1) *C-VF.* Close one active bin with size less than 1 if a such bin exists, otherwise use FF.
(2) *C-VB.* Close one active bin with size less than 1 if a such bin exists, otherwise use Best-Fit (BF).

They defined the opening rule in the following way: suppose a_i is a large item with size greater than 1/2. If it can be contained in a bin with size less than 1, it is called a B-item, otherwise it is called an L-item. The smallest bin that can contain a large item a_i is called an a_i-home-bin. Obviously, if a_i is an L-item, the size of the a_i-home-bin is 1. Their opening rule is, suppose the current item to be packed is a_i. If a_i is a B-item, open an a_i-home-bin and pack a_i into it. Otherwise, start a new bin of size 1 for a_i.

Burkard and Zhang proved that using this opening rule and the closing rule C-VB we will get a tight 1.7 performance bound for this algorithm for those cases where we have at least three open bins at the same time. This means that—taking into account Csirik and Johnson's result for classical bin packing—we need one more open bin for the variable-sized bin packing problem to have an algorithm with the same performance.

Chu and La [13] used a best-fit-like idea for variable-sized bin packing. They defined four algorithms: LARGEST OBJECT FIRST WITH LEAST ABSOLUTE WASTE (LFLAW), LARGEST OBJECT FIRST WITH LEAST RELATIVE WASTE (LRLRW), LEAST ABSOLUTE WASTE (LAW), and LEAST RELATIVE WASTE (LRW). Naturally, the first two are not online algorithms as they need a sorted list as input. They proved tight bounds for each algorithm, showing that

$$R^{\infty}_{LFLAW} = R^{\infty}_{LFLRW} = 2,\ R^{\infty}_{LAW} = 3 \text{ and } R^{\infty}_{LRW} = 2 + \ln 2$$

Zhang [14] introduced a relaxation on the online condition: Here we have all the information about the items, but we cannot preview the type of bin before it arrives. We must decide which items should be packed into the bin as it arrives. We will suppose that the largest item still fits in the smallest bin. In this case we have of course just one open bin, and we close it as we cannot pack more items into it. The goal is evidently to fill bins with the minimal total size. Zhang investigated the classical algorithms NF, FF, Next Fit Decreasing, and FFD and proved that each of them has an asymptotic worst-case ratio of 2. He also remarked that if there are at most l, $l \geq 1$, open bins at a time this ratio cannot be improved.

Dell'Olmo and Speranza [15] introduced a different relaxation on the variable-sized bin packing problem: They supposed that the bin sizes are extendible, that is, the total size of items packed into a single bin can exceed 1, if necessary. The goal is to minimize the total bin size, where the size of bin B_j is now given by $\max(c(B_j), s(B_j))$. The authors investigated two heuristics: Best Fit Decreasing (BFD) for the offline case and BF for the online case. They proved that BFD's performance bound is at most $2(2-\sqrt{2})$ and conjectured that the exact bound is 8/7. For the online case they proved a tight 5/4 bound for the BF algorithm and showed that no online algorithm can have better bound than 7/6.

Ye and Zhang [16] studied a similar problem. They supposed that we have m bins with different bin sizes. They called the set of bins a collection. We have to pack the items into these bins and overpacking is allowed. They distinguished two cases. In the first case, the largest item will fit into the smallest bin. They carefully analyzed the list scheduling algorithm and gave competitive ratios for each m and each collection. The ratios differ for an even number of bins and an odd number of bins. They were able to prove that

$$R_{LS}(m) = \begin{cases} 1 + \frac{m \cdot b_{\min}}{4\sum_{j=1}^{m} b_j} & \text{if } m \text{ is even} \\ 1 + \frac{(m^2-1)\cdot b_{\min}}{4m\sum_{j=1}^{m} b_j} & \text{if } m \text{ is odd} \end{cases}$$

where $b_{\min}$ is the smallest bin size. For equal bin size the ratio is 5/4 when m is even and $5/4 - 1/(4m^2)$ when m is odd. They presented an improved algorithm for $m = 2$ and $m = 3$.

In the second case, the largest item does not fit into the smallest bin. A lower bound for the overall competitive ratio is given for two bins.

Kang and Park [17] investigated the special case of variable-sized bin packing where we have (weak) divisibility conditions. Two types of conditions were looked at:

- where the sizes of the items are divisible,
- where the sizes of the bins are divisible, that is, $s(B_{j+1})$ exactly divides $s(B_j)$ for all $j = 1, 2, \ldots, k - 1$.

They also used a more general cost function: They assumed that the bin sizes are sorted in descending order, i.e., $b_1 \geq b_2 \geq \cdots \geq b_k$, and bins of size b_i have a cost c_i, where the unit size cost of each bin does not increase as the bin size increases, that is, the costs and the bin sizes satisfy $\frac{c_{i_1}}{b_{i_1}} \leq \frac{c_{i_2}}{b_{i_2}}$ for all $1 \leq i_1 \leq i_2 \leq k$. They studied iterative versions of the FFD and BFD rules. The ITERATIVE FIRST FIT DECREASING algorithm (IFFD) works as follows: We first pack all the items into the largest size bins using the FFD algorithm, and obtain a feasible solution. We then get another solution by repacking the items in the last bin of the solution into the next largest bin using the FFD rule. We obtain another feasible solution by continuing this procedure until we have repacked every item. In this way, we obtain feasible solutions for each type of bin. Then the best solution among them is selected as the final solution. The ITERATIVE BEST FIT DECREASING (IBFD) algorithm is similarly defined, using the BFD rule instead of the FFD rule in each step. They obtained a series of results for this algorithm:

- When L has divisible item sizes and bin sizes are divisible as well, there exists an optimal solution using k_1 or $k_1 - 1$ bins of type 1, where k_1 is the number of bins used in the first step of IFFD. This means that IFFD provides an optimal solution.
- For each L, if bin sizes are divisible, then $C(\mathit{IFFD}(L)) \leq \frac{11}{9} \cdot C(\mathit{OPT}(L)) + 5\frac{2}{9}$. Here $C(A(L))$ denotes the cost of packing list L by algorithm A. The bound of 11/9 is tight.
- The general case, that is, when we have no divisibility condition, for each L, $C(\mathit{IFFD}(L)) \leq \frac{3}{2} \cdot C(\mathit{OPT}(L)) + 1$. This bound is tight.

The same results apply to IBFD.

Xing [18] investigated a special case of variable-sized bin packing where we can have oversized items, that is, items with a size larger than the largest bin size. The bins cannot be overpacked, so we are free to divide up the oversized items such that the parts are no larger than the largest bin size. The problem is called *Bin Packing with Oversized Items,* (BPOS). Xing also defined a special objective function for this problem: If an item a_i is oversized and so packed into more than one bin, the extra bins do not contribute to the objective function. If we use bins $B_1, B_2, \ldots, B_m$ in the packing, then the objective function is

$$\sum_{i=1}^{m} s(B_i) - \sum_{i=1}^{n} (\lceil s(a_i) \rceil - 1)$$

Xing defined the following two-stage procedure: Let us delete size $\lfloor s(a_i) \rfloor$ from item a_i—now the size of the remaining part does not exceed the largest bin size. We can easily extend a heuristic algorithm of variable-sized bin packing to a heuristic of BPOS: first we pack $\lceil s(a_i) \rceil - 1$ unit-sized bins with unit parts of a_i and then the remaining part will be packed by the variable-sized bin packing heuristic. Let *TOPT(L)* denote the optimal value of the BPOS using a two-stage procedure. It can readily be seen that if there is only one bin size then *OPT*(L)=*TOPT*(L) for each list. If there are several bin sizes and the largest item is packed into l bins then

$$2 - \frac{1}{l+1} \leq R^{\infty}_{TOPT} \leq 2$$

and the ratio 2 is asymptotically tight.

Xing defined a modification of the FF algorithm for this problem and showed that its asymptotic worst-case ratio is 7/4. If there is a bin size between [2/3, 3/4], then the ratio is 3/2.

34.2.1 Number of Items

Langston [19] turned to the problem of maximizing the number of items packed into m available bins, where the bin sizes can be different. He analyzed the three heuristics given by Coffman et al. [20], and proved that both smallest-piece-first and first-fit-increasing have an asymptotic worst-case bound of 1/2. The best algorithm is the FFD* rule, for which Langston was able to prove that for all L and bin size-set B

$$n_{FFD} * (L, B) \geq \frac{2}{3} n_O(L, B) - \frac{2}{3}$$

He conjectured that the tight bound is 8/11—this is still an open question.

Friesen and Kuhl [21] gave the best known algorithm for maximizing the number of items in a given set of variable-sized bins. Their algorithm is an iterative version of the Best-2-Fit and FFD algorithms, and is similar to the compound algorithm given for the classical problem in Ref. [22]. This hybrid algorithm has a tight asymptotic worst-case ratio of 3/4.

Epstein and Favrholdt [23,24] investigated the online version of maximizing the number of items packed into variable-sized bins. They restricted the input sequences to be accommodating, that is, sequences that we know in advance every item can be packed by an optimal offline algorithm. They studied fair algorithms that reject an item only if the item does not fit in the empty space of any bin. They proved that any fair algorithm has a competitive ratio of at least 1/2, and BF has a performance bound of $n/(2n-1)$. They also showed that any fair, deterministic algorithm has a competitive ratio at most 2/3, and any fair, randomized algorithm has a competitive ratio at most 4/5.

34.3 Bin Covering

In the *packing* problems studied up to now, the goal was to partition a set of items into the *minimum* number of subsets such that, in every subset, the total size of the items does not exceed some upper bound. In a *covering* problem, the goal is to partition a set of items into the *maximum* number of subsets such that, in every subset, the total size of the items is always above some *lower* bound. Covering problems model a variety of situations encountered in business and in industry, from packing peach slices into tin cans so that each tin can contains at least its advertised net weight, to such complex problems as breaking up monopolies into smaller companies, each of which is large enough to be viable. Since covering problems can be viewed as a kind of inverse or dual version of the packing problem, they are sometimes called "dual-bin packing" problems in the literature. We note that the task of maximizing the number of items to be packed is sometimes also called dual-bin packing problem.

In the one-dimensional bin covering problem, the goal is to pack a list $L = (a_1, a_2, \ldots, a_n)$ into a maximum number of bins of size 1 such that the contents of each bin is at least one. Let R_A^∞ denote the asymptotic worst-case ratio of an approximation algorithm for bin covering ($A(L)$ and $OPT(L)$ denote the numbers of bins in the packing constructed by algorithm A and an optimization algorithm, respectively). It is defined by

$$R_A^n = \min\left\{ \frac{A(L)}{OPT(L)} \mid OPT(L) = n \right\}$$

and

$$R_A^\infty = \liminf_{n\to\infty} R_A^n$$

Since we are dealing with a maximization problem, the larger the worst-case ratio of an algorithm, the better the approximation algorithm will be. The bin covering problem was investigated for the first time in the Ph.D. thesis of Assmann [25] and in the journal article [2] by Assmann et al. [2]: They called it the *dual-bin packing problem.*

First of all one might adapt heuristics from the classical bin packing to the bin covering problem. This was indeed done by Assmann et al. [2]. They considered the following adaptation of NF, called DUAL NEXT FIT (DNF): DNF always has a single active bin. Newly arriving items a_i are packed into the active bin until the active bin is full (i.e., it has contents of at least one). Then the active bin is closed and another (empty) bin becomes the active bin. It is not difficult to show that DNF has performance

$$R^{\infty}_{DNF} = \frac{1}{2}$$

as all bins packed by DNF will have a content of less than 2, while for the lists

$$L_{4k} = \left(\underbrace{1-\varepsilon, \ldots, 1-\varepsilon}_{2k \text{ items}}, \underbrace{\varepsilon, \ldots, \varepsilon}_{2k \text{ items}} \right)$$

we get $\text{OPT}(L_{4k}) = 2k$ and $DNF(L_{4k}) = k$, if $\varepsilon < 1/2k$. Analogous modifications of FF, BF, and Harmonic *do not* improve this worst case ratio of 1/2 (e.g., modifying FF is useless, since after filling a bin, placing further items into it does not make sense). Actually, from the worst-case point of view, algorithm DNF is the best possible online bin covering algorithm, since Csirik and Totik [26] have proved that every online bin covering algorithm A satisfies the condition

$$R^{\infty}_{A} \leq \frac{1}{2}$$

A bettering of the performance ratio of 1/2 was achieved by Assmann et al. [2] by defining an *artificial upper bound* on the sum of the sizes of elements placed into the same bin. This upper bound can be regarded as the capacity of a bin and has some similarities with classical bin packing. However, after packing the items with a good heuristic for the classical bin packing problem, it might happen that in some of the bins the sum of item sizes is less than 1. Hence we have to use a second step to fill these bins. The algorithm based on the above observation is called FIRST FIT DECREASING WITH PARAMETER r (FFD_r) and proceeds as follows:

Phase I. ("Classical *FFD*")

(a) Presort the items in L such that $s(a_1) \geq s(a_2) \geq \cdots \geq s(a_n)$.
(b) While there is still an unpacked element, do the following: Let a_i be the first unpacked item and let B_j be the first (leftmost) unfilled bin with a current content less than or equal to $r - s(a_i)$. If such a bin exists, place a_i in B_j, otherwise open a new empty bin and pack a_i into this bin.

Phase II. (Repacking unfilled bins)

While there is more than one open nonfilled bin, remove an item from the rightmost such bin and add it to the leftmost one.

It is clear that after phase I there is no bin with content more than r. Furthermore, it follows from the definition in the first phase that adding an element from the rightmost open nonfilled bin will increase the total sum of sizes of elements in the leftmost bin to more than r. Lastly, the time complexity of FFD_r can be seen to be $O(n \log n)$. For this algorithm the following result holds [25]: for all r, $4/3 \leq r \leq 3/2$, and every lists L,

$$FFD_r(L) \geq \frac{2}{3}(OPT(L) - 1)$$

and the bound is tight.

Assmann et al. also suggested a further improvement by defining a pretty sophisticated algorithm called ITERATED LOWEST FIT DECREASING (ILFD). To define this heuristic we first consider the following problem: given the list L and a fixed number M of bins, what is the *maximum* possible value for the minimum bin level in a packing of L into M bins? From a good heuristic A for this problem we can derive a good approximation algorithm for the bin covering problem by iteratively applying this algorithm A. Let $A(L, M)$ stand for the minimum bin level in the packing of L generated by the heuristic A when the number of bins is fixed to be M. Now the algorithm iteratively applies A and proceeds as follows:

ITERATED "A"

Step 1. Let $UB = \left\lfloor \sum_{i=1}^{n} s(a_i) \right\rfloor$, $LB = 1$. (Clearly $LB \leq OPT(L) \leq UB$.)
Step 2. While $UB - LB > 1$ take $M = \lfloor (LB + UB)/2 \rfloor$ and apply heuristic A. If $A(L, M) > 1$ take $LB = M$, otherwise $UB = M$.

The resulting algorithm provides a feasible solution to the bin covering problem with *LB* bins. Clearly, the performance of this method depends on the choice of A. While the problem to be solved by A is closely related to multiprocessor scheduling problems, it seems natural to use for the heuristic A the Lowest Fit Decreasing (*LFD*) algorithm, as studied by Graham [27] and Deuermeyer et al. [28]. This algorithm proceeds as follows:

Step 1. Order L so that $s(a_1) \geq s(a_2) \geq \cdots \geq s(a_n)$ and start with M empty bins.
Step 2. While there is an unpacked item in L do the following: let a_i be the first unpacked item and let B_j be the bin with a minimum level (in case of ties, choose the rightmost one). Put a_i into B_j.

It is not difficult to verify that the time complexity of *ILFD* is $O(n \log^2 n)$. Furthermore, one can prove the following result [2]:

$$R^{\infty}_{ILFD} = \frac{3}{4}$$

Csirik et al. [29] achieved these bounds via simpler algorithms. Their first algorithm, called SIMPLE (SI), first sorts the items in descending order, and then packs the prefix of the list by NF into the next bin until the content becomes larger than 1 when the piece following the prefix is packed. It then fills the bin with a postfix, that is, packs the smallest items as long as the content of the bin is less than 1. For this algorithm

$$R^{\infty}_{SI} = \frac{2}{3}$$

The second algorithm, IMPROVED SIMPLE (ISI) is a bit more complicated. It divides the list L into three sublists:

- $s(a_1) \geq s(a_2) \geq \cdots \geq s(a_p) \geq 1/2$(X-sublist),
- $1/2 > s(a_{p+1}) \geq s(a_{p+2}) \geq \cdots \geq s(a_r) \geq 1/3$(Y-sublist),
- $s(a_{r+1}) \geq s(a_{r+2}) \geq \cdots \geq s(a_n)$(Z-sublist).

Now, ISI proceeds in two phases. In phase 1, if $s(a_1) \geq s(a_{p+1}) + s(a_{p+2})$, it then packs a_1 into an empty bin, otherwise packs a_{p+1} and a_{p+2}. Then it fills the just opened bin with elements from the end of the Z-sublist, that is, with $a_n, a_{n-1}, \ldots$ until the content of the bin is larger than 1, removes the packed elements, and then repeats the packing until $X \cup Y$ or Z is empty. In the second phase, if $X \cup Y$ is empty, it packs the remaining elements from the Z-sublist by NF. Otherwise, if Z is empty, it packs the remaining elements from the X-sublist by two, from the Y-sublist by three. The authors subsequently proved that

$$R^{\infty}_{ISI} = \frac{3}{4}$$

The time complexity of both algorithms here is $O(n \log n)$.

Coffman et al. [30] analyzed the bin covering problem when the list has divisible item sizes. They proved that the Next Fit Decreasing rule gives an optimal packing when the list is strongly divisible. They also proved that if the list is divisible, the Iterated Lowest Fit Decreasing rule gives an optimal packing as well.

Deuermeyer et al. [28] studied the problem of machine covering. Here we have m machines (bins) and we want to pack the list so as to maximize the minimum load for the machines. This means in bin covering terms that we would like to maximize the bin sizes such that the packing of the list will cover all the bins.

They investigated the performance of the Largest Processing Time (LPT) heuristic and proved that its performance is not larger than 3/4. The tight ratio of this method was found by Csirik et al. [31] to be $\frac{3m-1}{4m-2}$.

Azar and Epstein [32] investigated the online version of machine covering. They proved that there is a randomized $O(\sqrt{m}\log m)$ competitive algorithm, and any randomized algorithm is at least $\Omega(\sqrt{m})$ competitive. This is in marked contrast to the competitive ratio of the best possible deterministic algorithm, which is m. For the parametric version of this problem, where the sizes may vary up to a factor F they showed that there is a randomized $O(\log F)$ competitive algorithm. For the same problem where the optimal value is known in advance they offered a deterministic $2 - \frac{1}{m}$ competitive algorithm.

Woeginger and Zhang [33] considered bin covering with variable sized bins: there are several types $B_1, \dots, B_k$ of bins with sizes $1 = b_1 = s(B_1) > b_2 = s(B_2) > \cdots > b_k = s(B_k)$; there is an infinite supply of bins of each size. We will denote the set of feasible bin sizes here by **B** as well. The problem is to cover, with a given list of items $a_i \in [0, 1]$, a set of bins with the *largest total size*. Formally, let $OPT(L, \mathbf{B})$ and $A(L, \mathbf{B})$ denote, respectively, the total size of the bin cover produced by an optimum algorithm and the total size of the bin cover produced by an approximation algorithm A for an input list L and a set **B** of bin sizes. The asymptotic worst-case $R^{\infty}_{A,\mathbf{B}}$ of algorithm A for the set **B** is defined as

$$R^{\infty}_{A,\mathbf{B}} = \lim_{s\to\infty} \inf_{L} \left\{ \frac{A(L, \mathbf{B})}{OPT(L, \mathbf{B})} \mid OPT(L, \mathbf{B}) \geq s \right\}$$

Woeginger and Zhang [33] determined, for each finite set of bin sizes, the worst-case ratio of the best possible online covering algorithm. Let k_1 denote the number of bin sizes in **B** that are strictly greater than 1/2. Define

$$q(\mathbf{B}) = \max \left\{ \frac{b_j}{b_{j+1}} : 1 \leq j \leq k_1 - 1 \right\} \cup \{2b_{k_1}\}$$

Note that $q(\mathbf{B}) > 1$. Finally, define

$$r(\mathbf{B}) = \frac{1}{q(\mathbf{B})}$$

Woeginger and Zhang proved that for every set **B** of bin sizes, there exists an online approximation algorithm A for variable-sized bin covering with asymptotic worst-case ratio $r(\mathbf{B})$. For every set **B** this result is the best possible.

Epstein [34] gave similar results for the parametric case, that is, where each item size is less than or equal to $1/m$, where m is a positive integer. Similar to Ref. [33], she defined the number $r(\mathbf{B}, m)$ in the following way. For each $1 \leq i \leq k$, let $b_{i,j} = b_i/j$ and let $B_i(m)$ be the set of fractions of b_i between sizes $1/(2m)$ and 1/m, that is, $\mathbf{B}_i(m) = \{b_{i,j} \mid 1 \leq j \leq 2m\} \cap [1/(2m), 1/m]$. We define a new set of bins by $\mathbf{C}(m) = \cup_{1\leq i\leq k}\mathbf{B}_i(m)$. Enumerate the sizes of numbers in $\mathbf{C}(m)$, $\mathbf{C}(m) = \{c_1, \dots, c_l\}$, where $1/m = c_1 > c_2 > \cdots > c_l = 1/(2m)$. For every element in $\mathbf{C}(m)$, recall an original bin size that caused it to be inserted into $\mathbf{C}(m)$. For c_i, let $b(c_i)$ be the smallest b_j such that there exists an integer y that satisfies $yc_i = b_j$. Now define

$$q(\mathbf{B}, m) = \max \left\{ \frac{c_i}{c_{i+1}} \mid 1 \leq i \leq l - 1 \right\}$$

Note that $1 + 1/m \geq q(\mathbf{B}, m) > 1$. Finally, define

$$r(\mathbf{B}, m) = \frac{1}{q(\mathbf{B}, m)}$$

For every finite set of bins and integer $m \geq 1$, Epstein gave a deterministic algorithm with competitive ratio $r(\mathbf{B}, m)$. She also proved that this is the best possible for any online randomized algorithms.

Zhang [35] defined a special variable-sized bin covering problem which he called the *bin-batching* problem: in this problem, in addition to a list L, a parameter $0 \leq t \leq 1$ is also given. We are asked to group the item into batches. If a batch B has a total content $c(B)$, the gain of this batch is defined as

$$g(B) = \begin{cases} 0 & \text{if } c(B) < 1 - t \\ c(B) & \text{if } 1 - t \leq c(B) \leq 1 \\ 1 & \text{if } c(B) > 1 \end{cases}$$

The goal is to make the total gain as large as possible. We may note that this problem reduces to the bin covering problem when $t = 0$. In bin batching all bin sizes in $[1 - t, 1]$ are allowed, that is, we allow underpacking of bins up to a level of $1 - t$. It is clear that the problem is simple if $t \geq 1/2$, so we may assume $t < 1/2$. Zhang proved that every online algorithm for the bin-batching problem has an asymptotic worst-case ratio of at most $1/(2 - 2t)$, and that the following simple algorithm—called SA—will meet this bound: assume that the currently incoming item is a_i; if $s(a_i) \geq 1 - t$, put it into an empty bin with size $s(a_i)$ and close this bin immediately. If $s(a_i) < 1 - t$, we apply the NF algorithm with bin size 1.

Woeginger and Zhang [33] looked at the special problem of having only two different bin sizes (this was done by Csirik [6] for the classical variable-sized problem). They proved that, for a set $\mathbf{B} = \{1, b\}$ with two bin sizes, there exists an online approximation algorithm A for the bin covering problem with the best possible asymptotic worst-case ratio

$$R^{\infty}_{A,\mathbf{B}} = \begin{cases} \frac{1}{2} & \text{if } 0 < b \leq \frac{1}{2} \\ b & \text{if } \frac{1}{2} \leq b \leq \frac{1}{\sqrt{2}} \\ \frac{1}{2b} & \text{if } \frac{1}{\sqrt{2}} \leq b < 1 \end{cases}$$

For $\mathbf{B} = \{1, 1/\sqrt{2}\}$, the asymptotic worst-case ratio is $1/\sqrt{2}$. This is the best ratio that can be achieved with two bins.

34.4 Conclusion

In this chapter we conclude our survey of variants of the classical bin packing problem. The main topics covered include the variable-sized bin packing problem and the bin covering problem. Both problems have generated tremendous interests in the past and promise to have more results in the future.

References

[1] Friesen, D. K. and Langston, M. A., Variable sized bin packing, *SIAM J. Comput.*, 15, 222, 1986.

[2] Assmann, S. F., Johnson, D. S., Kleitman, D. J., and Leung, J. Y.-T., On a dual version of the one-dimensional bin packing problem, *J. Algorithms*, 5, 502, 1984.

[3] Friesen, D. K. and Langston, M. A., A storage-selection problem, *Inf. Proc. Lett.*, 18, 295, 1984.

[4] Kinnersley, N. G. and Langston, M. A., Online variable-sized bin packing, *Disc. Appl. Math.*, 22, 143, 1988.

[5] Zhang, G., Worst-case analysis of the FFH algorithm for online variable-sized bin packing, *Computing*, 56, 165, 1996.

[6] Csirik, J., An on-line algorithm for variable-sized bin packing, *Acta Informatica*, 26, 697, 1989.

[7] Seiden, S. S., An optimal online algorithm for bounded space variable-sized bin packing, *SIAM J. Disc. Math.*, 458, 2001.

[8] Seiden, S. S., van Stee, R., and Epstein, L., New bounds for variable-sized online bin packing, *SIAM J. Comput.*, 33, 455, 2003.

[9] Seiden, S. S., An optimal online algorithm for bounded space variable-sized bin packing, *Proc. ICALP*, 2000, p. 283.

[10] Epstein, L., Seiden, S., and van Stee, R., New bounds for variable-sized and resource augmented online bin packing, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 2380, Springer, Berlin, 2002, p. 306.

[11] Burkard, R. E. and Zhang, G., Bounded space on-line variable-sized bin packing, *Acta Cybernetica*, 13, 63, 1997.

[12] Csirik, J. and Johnson, D. S., Bounded space on-line bin packing: best is better than first, *Algorithmica*, 31, 115, 2001.

[13] Chu, C. and La, R., Variable-sized bin packing: Tight absolute worst-case performance ratios for four approximation algorithms, *SIAM J. Computing*, 30, 2069, 2000.

[14] Zhang, G., A new version of on-line variable-sized bin packing, *Disc. Appl. Math.*, 72, 193, 1997.

[15] Dell'Olmo, P. and Speranza, M. G., Approximation algorithms for partitioning small items in unequal bins to minimize the total size, *Disc. Appl. Math.*, 94, 181, 1999.

[16] Ye, D. and Zhang, G., On-line extensible bin packing with unequal bin sizes, *Proc. WAOA*, Lecture Notes in Computer Science, Vol. 2909, Springer, Berlin, 2004, p. 235.

[17] Kang, J. and Park, S., Algorithms for the variable sized bin packing problem, *Eur. J. Oper. Res.*, 147(2), 365, 2003.

[18] Xing, W., A bin packing problem with over-sized items, *Oper. Res. Lett.*, 30, 83, 2002.

[19] Langston, M. A., Performance of heuristics for a computer resource allocation problem, *SIAM J. Alg. Disc. Meth.*, 5, 154, 1984.

[20] Coffman, Jr., E. G., Leung, J. Y.-T., and Ting, D. W., Bin packing: maximizing the number of pieces packed, *Acta Informatica*, 9, 263, 1978.

[21] Friesen, D. K. and Kuhl, F. S., Analysis of a hybrid algorithm for packing unequal bins, *SIAM J. Comput.*, 17, 23, 1988.

[22] Friesen, D. K. and Langston, M. A., Analysis of a compound bin-packing algorithm, *SIAM J. Disc. Math.*, 4, 61, 1991.

[23] Epstein, L. and Favrholdt, L. M., On-line maximizing the number of items packed in variable-sized bins, *Proc. COCOON*, 2002, p. 467.

[24] Epstein, L. and Favrholdt, L. M., On-line maximizing the number of items packed in variable-sized bins, *Acta Cybern.*, 16, 57, 2003.

[25] Assmann, S. F., Problems in Discrete Applied Mathematics, Ph.D. thesis, Department of Mathematics, MIT, Cambridge, MA, 1983.

[26] Csirik, J. and Totik, V., On-line algorithms for a dual version of bin packing, *Disc. Appl. Math.*, 21, 163, 1988.

[27] Graham, R. L., Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.*, 17, 416, 1969.

[28] Deuermeyer, B. L., Friesen, D. K., and Langston, M. A., Scheduling to maximize the minimum processor finish time in a multiprocessor system, *SIAM J. Alg. Disc. Meth.*, 3, 190, 1982.

[29] Csirik, J., Frenk, J. B. G., Labbé, M., and Zhang, S., Two simple algorithms for bincovering, *Acta Cybern.*, 14, 13, 1997.

[30] Coffman, E. G., Jr., Garey, M. R., and Johnson, D. S., Bin packing with divisible item sizes, *J. Complexity*, 3, 405, 1987.

[31] Csirik, J., Kellerer, H., and Woeginger, G., The exact LPT-bound for maximizing the minimum completion time, *Oper. Res. Lett.*, 11, 281, 1992.

[32] Azar, Y. and Epstein, L., On-line machine covering, *Eur. Symp. on Algorithms*, Lecture Notes in Computer Science, Vol. 1284, Springer, Berlin, 1997, p. 23.

[33] Woeginger, G. J. and Zhang, G., Optimal on-line algorithms for variable-sized bin covering, *Oper. Res. Lett.*, 25, 47, 1999.

[34] Epstein, L., On-line variable sized covering, *Inf. Comp.*, 171, 294, 2001.

[35] Zhang, G., An on-line bin batching problem, *Disc. Appl. Math.*, 108, 329, 2001.

35

Multidimensional Packing Problems

Leah Epstein
University of Haifa

Rob van Stee
University of Karlsruhe

35.1 Introduction

There are several ways to generalize the bin packing problem to more dimensions. We consider two- and three-dimensional strip packing, and bin packing in dimensions two and higher. Finally we consider vector packing and several other variations.

In the most common two-dimensional version, the items are rectangles or squares, and the bins are unit squares. In the strip packing problem, instead of bins, we are given a strip of width 1 and unbounded height. In higher dimensions, the rectangles are replaced by boxes (or hyperboxes), the squares by cubes (or hypercubes), and the unit square by a unit cube (or hypercube of the relevant dimension). Strip packing becomes column packing.

A striking difference between one-dimensional bin packing and its multidimensional generalizations is that while for one-dimensional bin packing, offline algorithms clearly outperform online algorithms, this is not always the case in more dimensions. There are several cases where an online algorithm was at one point the best known approximation algorithm, or remains the best known approximation until today. Most likely, this simply reflects the fact that we do not understand the multidimensional case as well as the one-dimensional case. In contrast, some results simply cannot be generalized. For instance, we now know that there cannot be an asymptotic polynomial-time approximation scheme (APTAS) for two-dimensional bin packing [1], or for two-dimensional vector packing [2].

An important special case in multidimensional bin and strip packing is the case where (hyper)cubes need to be packed. For this case, better results are known than for the general case. In particular, the offline version of this problem admits an APTAS [1,3].

As is the case for one-dimensional bin packing, most attention has gone to the asymptotic worst-case ratio, but in the course of this chapter we will encounter some results on the absolute ratio as well.

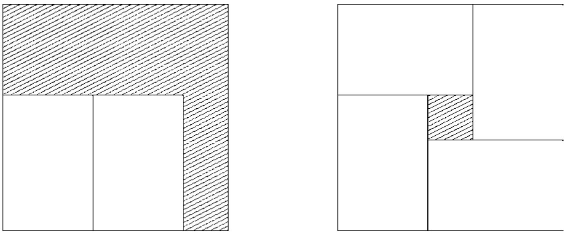


FIGURE 35.1 A comparison between the oriented and the rotatable models.

When packing of rectangle or boxes is considered, there are several ways to define the problem. In the oriented problem, items have a fixed orientation, and cannot be rotated. In the rotatable (or nonoriented) version, an item can be rotated and placed in any position such that its sides are parallel to the sides of the bin. Finally there are mixed versions where items can be rotated in certain directions, but not all directions. One such three-dimensional model where items can be rotated to the left or to the right but the top and bottom must remain such as the "z-oriented" packing studied by Miyazawa and Wakabayashi [4,5], also known as the "This Side Up" problem [6].

An illustration of the difference between the two problems is given in Figure 35.1. In this figure we see packings of rectangles of sides $\frac{3}{5}$ and $\frac{2}{5}$. If the rectangles are oriented so that their height is $\frac{3}{5}$ and cannot be rotated, we can pack at most two such items in one bin. However, if rotation is allowed, we can pack as much as four such rectangles together in one bin.

This chapter is organized as follows. We begin by presenting the algorithm Next Fit Decreasing Height, which is a fundamental algorithm for two-dimensional packing problems, in Section 35.2. We then discuss results on multidimensional packing problems, in order of increasing dimension. That is, we start with strip packing in Section 35.3 and move to two-dimensional bin packing in Section 35.4. We then discuss column packing in Section 35.5 and three- and more dimensional bin packing in Section 35.6. Finally, we mention results on vector packing in Section 35.7 and discuss several variations on multidimensional packing in Section 35.8.

35.2 Next Fit Decreasing Height

In 1968, Meir and Moser [7] introduced an algorithm for packing d-dimensional cubes into a d-dimensional hyperbox, which they called Next Fit Decreasing (NFD). This algorithm sorts the cubes by decreasing volume and packs them into layers. The authors show that if the sides of the cubes are denoted by $x_1, x_2, \ldots,$ and they are packed into a hyperbox of sides $a_1, \ldots, a_d$, where $x_1 \leq a_i$ for $i = 1, \ldots, d$, then the cubes can be packed into the hyperbox as long as their total volume is at most $x_1^d + \prod_{i=1}^{d}(a_i - x_1)$.

For $d = 2$ (packing squares into a rectangle), the algorithm works as follows. The largest square is put in the bottom left corner of the rectangle. The height of the first layer is equal to the side of this square. The next squares are put in this layer, next to each other and touching each other and the bottom of the layer, until one does not fit. At this point we define a new layer above the first layer, with height equal to the side of the first square packed into it. This continues until all squares are packed, or there is not enough room to pack some item (it does not fit into the current layer, and the last layer that is left is either empty or not high enough).

This algorithm (for two dimensions) was extended to an algorithm for packing rectangles into a rectangle (or a strip) by Coffman et al. [8], which was called Next Fit Decreasing Height (NFDH). It sorts the rectangles by decreasing height and then packs them as above. They showed that if this algorithm is applied to pack rectangles into a strip (of unbounded height), then the height used to pack the rectangles is at most twice the optimal height, plus an additive constant that is equal to the height of the highest rectangle. (Thus its absolute worst-case ratio is 3.)

The proof is quite straightforward. In each level, there may be wasted space to the right of the rightmost item, and above all items except the first. The height of a level is the height of the first item in it. This item did not fit on the previous level. This implies that the total area of the items in level i plus the first item in level $i + 1$ is at least the height of level $i + 1$ (since the width of the strip is 1). (If we move all items in level i up to level $i + 1$, and shift the first item in level $i + 1$ to the right, then level $i + 1$ is entirely covered by items.) Adding up the heights of all levels, this is upper bounded by twice the area of the packed items plus the height of the first level. This explains the performance bound including the additive constant, since the total area is an obvious lower bound for the optimal height.

This fundamental algorithm was used in many later papers as a subroutine. It works especially well when all rectangles are guaranteed to have a small width (relative to the width of the strip), and this property was, for instance, used by Kenyon and Rémila [9] in their approximation scheme for strip packing.

Meir and Moser [7] also showed the following important result in the same paper:

Theorem 35.1

Any set of rectangles with sides at most x and total area A can be packed into any rectangle of size $a \times b$ if $a \geq x$ and $ab \geq 2A + a^2/8$. This result is best possible.

For packing rectangles into a unit square, this result states that any set of rectangles of total area at most 7/16 (and sides not larger than 1) can be packed into a unit square.

35.3 Strip Packing

35.3.1 Online Results

Baker and Schwartz [10] were the first to study two-dimensional online strip packing. They introduced a class of algorithms called *shelf algorithms.* A shelf algorithm uses a one-dimensional bin packing algorithm A and a parameter $\alpha \in (0, 1)$. Items are classified by height: an item is in class s if its height is in the interval $(\alpha^{s-1}, \alpha^s]$. Each class is packed in separate *shelves*, where we use A to fill a shelf and open a new shelf when necessary. Note that the algorithm A is not necessarily online. See Figure 35.2 for an illustration of a shelf algorithm.

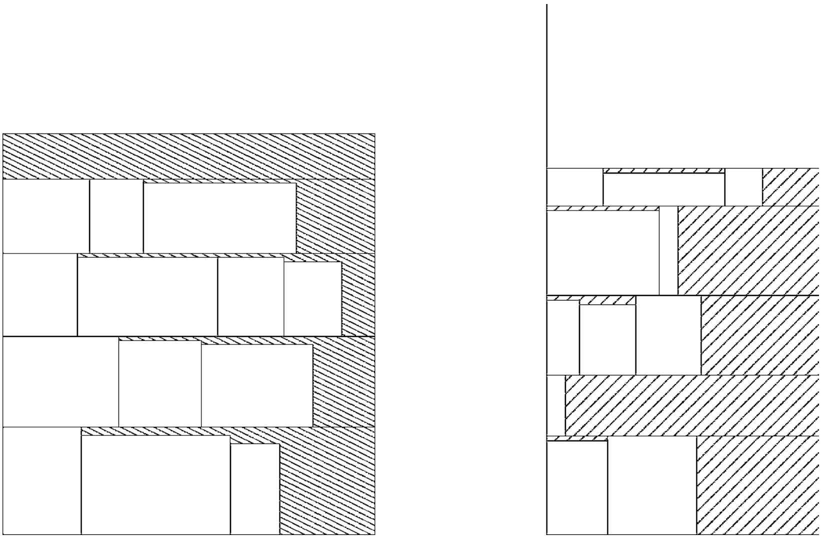


FIGURE 35.2 An illustration of a packing of NFDH (left) and of a shelf packing algorithm (right).

Baker and Schwartz showed that the algorithm FIRST FIT SHELF, which uses FIRST FIT as a subroutine, has an asymptotic performance ratio arbitrarily close to 1.7. Csirik and Woeginger [11] showed that by using HARMONIC as a subroutine, it is possible to achieve an asymptotic performance ratio arbitrarily close to $h_\infty \approx 1.69103$. Moreover, they show that any shelf algorithm, online or offline, has a performance ratio of at least h_∞. The idea of the lower bound is that items are given that could be combined nicely next to each other, but which end up in different height classes and are therefore packed in separate shelves. So basically, the best thing one can do is to use a bounded space algorithm (which has a constant number of simultaneously active bins) like HARMONIC as the subroutine. Finally, they mention that from the one-dimensional lower bound of van Vliet [12], together with the insights of Baker et al. [13], a general lower bound for online algorithms of 1.5401 is implied. It remains an open problem how to improve the upper bound of Csirik and Woeginger. It does not seem easy to find a good online algorithm that does not use shelves. As for the absolute performance ratio, Brown et al. [14] showed a lower bound of 2 for any algorithm. They also show some lower bounds for algorithms that may sort the items.

35.3.2 Offline Results

The strip packing problem was introduced in 1980 by Baker et al. [15]. They developed the first offline approximation algorithms for this problem, and give an upper bound of 3 on the absolute performance ratio. This bound was later improved to 2 independently by Schiermeyer [16] and by Steinberg [17], using different approaches. In the same issue of *SIAM Journal on Computing*, Coffman et al. [8] showed that NFDH has an asymptotic performance ratio of 2, First Fit Decreasing Height (FFDH) achieves a value of 1.7, and an algorithm called Split-Fit has 3/2. Also in 1980, Sleator [18] gave an algorithm with asymptotic performance ratio of 2.5, but absolute performance ratio of 2, which is better than that of Split-Fit, which has 3. In 1981, Baker et al. [13] gave an offline algorithm with asymptotic worst-case ratio of 5/4. Finally, Kenyon and Rémila [9] designed an asymptotic fully polynomial-time approximation scheme. This scheme uses some nice ideas, which we describe below.

Fractional strip packing. A fractional strip packing of L is a packing of any list L' obtained from L by subdividing some of its rectangles by horizontal cuts: each rectangle (w_i, h_i) is replaced by a sequence of rectangles $(w_i, h_i^1), (w_i, h_i^2), \ldots, (w_i, h_i^{k_i})$ such that $\sum_{j=1}^{k_i} h_i^j = h_i$.

In the case that L contains only items of m distinct widths in $(\varepsilon', 1]$, where $\varepsilon' > 0$ is some constant, it is possible to find a fractional strip packing of L that is within one of the optimal fractional strip packing $\textsc{fsp}(L)$ in polynomial time. Moreover, it is possible to turn this packing into a regular strip packing at the loss of only an additive constant $2m$. Denote the height of the optimal strip packing for L by $\textsc{opt}(L)$. We conclude that we found a packing with height at most $\textsc{fsp}(L) + 1 + 2m \leq \textsc{opt}(L) + 2m + 1$.

Modified NFDH. This is a method for adding narrow items (items of width at most ε') to a packing of wide items such as described above. Such a packing leaves empty rectangles on the right-hand side of the strip. Each of these rectangles is packed with narrow items using NFDH (starting with the highest narrow item in the first rectangle). When all rectangles have been used, the remaining items (if any) are packed above the packing using NFDH on the entire width of the strip.

Grouping and rounding. This method is a variation on the linear rounding defined by Fernandez de la Vega and Lueker [19]. It works as follows.

We stack up the rectangles of L by order of nonincreasing widths to obtain a left-justified stack of total height $h(L)$. We define $m-1$ threshold rectangles, where a rectangle is a threshold rectangle if its interior or lower boundary intersects some line $y = ih(L)/m$ for some $i \in \{1, \ldots, m-1\}$. We cut these threshold rectangles along the lines $y = ih(L)/m$. This creates m groups of items that have height exactly $h(L)/m$.

First, the widths of the rectangles in the first group are rounded up to 1, and the widths of the rectangles in each subsequent group are rounded up to the widest width in that group. This defines L_+.

Second, the widths of the rectangles in each group are rounded down to the widest width of the next group (down to 0 for the last group). This defines L_-.

It is easy to find a strip packing for L_- using a reduction to fractional strip packing. Moreover, it can be seen that the stack associated with L_+ is exactly the union of a bottom part of width 1 and height $h(L)/m$ and the stack associated with L_-. Thus $\text{FSP}(L) \leq \text{FSP}(L_+) = \text{FSP}(L_-) + h(L)/m$.

Partial ordering. We say that $L \leq L'$ if the stack associated to L (used for the grouping above), viewed as a region of the plane, is contained in the stack associated to L'. Note that $L \leq L'$ implies that $\text{FSP}(L) \leq \text{FSP}(L')$. As an example, in the grouping above we have $L_- \leq L \leq L_+$.

35.3.3 Rotations

The upper bound of 2 of NFDH and Bottom Leftmost Decreasing Width (BLDW) remain valid if orthogonal rotations are allowed, since the proofs use only area arguments. Miyazawa and Wakabayashi [5] presented an algorithm with asymptotic approximation ratio of 1.613. Epstein and van Stee [20] improved this to 3/2 using a simpler algorithm. This algorithm packs items that are wider and higher than 1/2 optimally, and packs remaining items first next to this packing (where possible) and finally on top of this packing. In this way, the resulting packing is either optimal, or almost all heights with a width of 2/3 is occupied by items. Finally, an asymptotic fully polynomial-time approximation scheme was given by Jansen and van Stee [21].

35.4 Two-Dimensional Bin Packing

We saw in section 35.3.1 that we can use a one-dimensional bin packing algorithm as a subroutine for a strip packing algorithm, basically without a loss in (asymptotic) performance ratio. Similarly, a two-dimensional bin packing algorithm can be used as a subroutine to create a three-dimensional strip packing algorithm, and this also holds for higher dimensions.

In contrast, a d-dimensional strip packing algorithm can also be used to create a d-dimensional bin packing algorithm at a cost of a factor of 2 in the performance ratio. The idea is to cut the packing generated by the strip packing algorithm into pieces of unit height. For each piece we do the following. Items that are completely contained in the piece are put together in one bin. Items that are partially in the next piece are put together in a second bin (see Figure 35.3).

Assume we have a guarantee of R on the asymptotic performance ratio of the strip packing algorithm. Then this method gives us $2R \cdot \text{OPT}(L) + C$ bins for an input L, where $\text{OPT}(L)$ is the height of the optimal

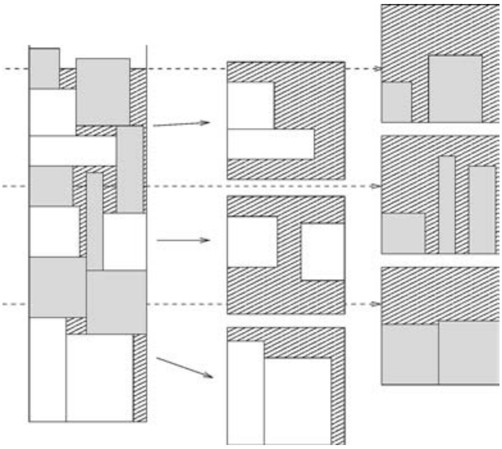


FIGURE 35.3 Converting a packing in a strip into a packing in bins.

strip packing. In contrast, there cannot be a *bin* packing into less than $\text{OPT}(L)$ bins, because this packing could be trivially turned into a strip packing of height less than $\text{OPT}(L)$. This explains the factor of two loss.

35.4.1 Online Results

Coppersmith and Raghavan [22] were the first to study the online version of this problem. They gave an online algorithm with asymptotic performance ratio of 3.25 for $d = 2$ (and 6.25 for $d = 3$). This result was improved by Csirik et al. [23], who presented an algorithm with performance ratio 3.0625. In the same year, Csirik and van Vliet [24] showed an online bin packing algorithm for arbitrary dimensions, which achieves a performance ratio of h_∞^d, where d is the dimension. Note that already for $d = 2$, this improves over the previous result, since $h_\infty^2 \approx 2.85958$ (see also Ref. [25] for $d = 2, 3$). Finally, Seiden and van Stee [26] gave an algorithm with ratio 2.66013 for two-dimensional bin packing.

Epstein and van Stee [20] introduced a new technique for packing small multidimensional items online, enabling them to achieve the asymptotic performance ratio of h_∞^d [24] using only bounded space.

Galambos [27] was the first to give a lower bound for this problem, which was higher than the best known lower bound for one-dimensional bin packing. His bound was 1.6. This was later successively improved to 1.808 by Galambos and van Vliet [28], 1.851 by van Vliet [29], and finally to 1.907 by Blitz et al. [30]. The gap between the upper and lower bounds remains relatively large to this day, and it is unclear how to improve either of them significantly.

An interesting special case is where all items are squares. Coppersmith and Raghavan [22] showed that their algorithm has an asymptotic performance ratio of 2.6875 for this case, and gave a lower bound of 4/3. This lower bound actually holds for the more general problem of packing hypercubes. Seiden and van Stee [26] showed that the algorithm HARMONIC × HARMONIC, which uses the HARMONIC algorithm to find slices for items and then uses the HARMONIC algorithm again to find bins for slices, has an asymptotic performance ratio of at most 2.43828. They gave a lower bound of 1.62176 for any online algorithm, and also showed a lower bound of 2.28229 for bounded space algorithms using the same instances.

Epstein and van Stee [31] give an algorithm with asymptotic performance ratio of at most 2.24437, and improved the lower bound to 1.6406. Here too, the gap between the lower and the upper bounds remains disappointingly large. Finally, the same authors [32] give bounds for the performance of their optimal bounded space algorithm from Ref. [20], showing that its performance ratio lies between 2.3638 and 2.3692.

35.4.2 Offline Results

As mentioned at the start of this chapter, Bansal and Sviridenko [1] proved that the two-dimensional bin packing problem is APX-hard. Thus, there cannot be an asymptotic polynomial-time approximation scheme for this problem.

Chung et al. [33] were the first to give an approximation algorithm for this problem. It has an asymptotic approximation ratio of 2.125. As mentioned above, the APTAS for strip packing by Kenyon and Rémila implies a $(2 + \varepsilon)$-approximation for any $\varepsilon > 0$. In 2002, Caprara [34] gave an h_∞-approximation.

Leung et al. [35] proved that the special case of packing squares into squares is still NP-hard (for general two-dimensional bin packing, this follows immediately from the one-dimensional case). Ferreira et al. [36] gave a 1.988-approximation for this problem, which uses as a subroutine an optimal algorithm for packing items with sides larger than 1/3. They conjecture that packing items with sides larger than 1/4 is already NP-hard. Independently of each other, Kohayakawa et al. [37] and Seiden and van Stee [26] gave a $(14/9 + \varepsilon)$-approximation $(1.5555\ldots + \varepsilon)$. However, the first result is more general in that it actually gives a $(2 - (2/3)^d + \varepsilon)$-approximation for packing d-dimensional hypercubes. The idea of both these algorithms is to find an optimal packing for large items (items with sides larger than ε) and to add the small items to this packing. Specifically, any bins in the optimal packing which contain only a single item with sides larger than 1/2 are filled with small items using the algorithm NFD from Meir and Moser (see Section 35.2). It is shown that all other bins are already "reasonably full," leading to the approximation guarantee.

In the same year, Caprara [34] gave an algorithm with performance ratio in the interval (1.490, 1.507) provided a certain conjecture holds. Two years later, Epstein and van Stee [38] gave a $(16/11 + \varepsilon)$-approximation $(1.4545 \ldots + \varepsilon)$. Simultaneously and independently of each other, Bansal and Sviridenko [1], and Correa and Kenyon [3] presented an asymptotic polynomial-time approximation scheme for this problem, which also works for the more general problem of packing hypercubes.

Recently, Bansal et al. [39] showed a special case which admits an APTAS. This is a rectangle packing, where the packing of each bin must be possible to achieve using guillotine cuts only. That is a sequence of edge to edge cuts, parallel to the edges of the bin. Even more special cases, where the number of stages in the sequence of guillotine cuts is limited, were studied by Caprara et al. [40]. They designed an APTAS for the two-stage problem. Note the shelf packing described above actually uses two stages of guillotine cuts. Kenyon and Rémila [9] point out that their approximation scheme uses five stages of guillotine cuts.

As regards the absolute performance ratio, Zhang [41] gave an approximation algorithm with absolute worst-case ratio of 3 for two-dimensional bin packing. Van Stee [42] gave an absolute 2-approximation for the special case where squares need to be packed, which is optimal by the result of Leung et al. [35].

35.4.3 Resource Augmentation

Since there cannot be an approximation scheme for general two-dimensional bin packing, several authors have looked at the possibility of resource augmentation, that is, giving the approximation algorithm slightly larger bins than the offline algorithm that it is compared with. Correa and Kenyon [3] give a dual polynomial-time approximation scheme. That is, they give a polynomial-time algorithm to pack rectangles into the k bins of size $1 + \varepsilon$, where these rectangles cannot be packed in less than k bins of size 1. Bansal and Sviridenko [43] showed that it is possible to achieve this even if the size of the bin is relaxed in one dimension only.

35.4.4 Rotations

For the case where rotations are allowed, Epstein [44] showed an online algorithm with asymptotic performance ratio of 2.45. The online problem was studied before by Fujita and Hada [45]. They presented two online algorithms and claimed asymptotic performance ratios of at most 2.6112 and 2.56411. Epstein [44] mentioned that the proof in Ref. [45] only shows that the first algorithm has an asymptotic performance ratio of at most 2.63889 and that the proof of the second algorithm is incomplete.

Two years later, Miyazawa and Wakabayashi [5] gave an offline algorithm with asymptotic performance ratio of 2.64. Epstein and van Stee [20] gave the best known result to date, an approximation algorithm with asymptotic performance ratio 2.25. It divides the items into types and combines them into bins such that in almost all bins, an area of 4/9 is occupied. Correa [46] adapted the dual polynomial-time approximation scheme from Ref. [3] to rotatable items.

35.5 Column (Three-Dimensional Strip) Packing

35.5.1 Online and Offline Results

Li and Cheng [47] were the first to consider this problem. In their paper [47] from 1990, they showed that three-dimensional versions of NFDH and FFDH have unbounded worst-case ratio. They gave several approximation algorithms, the best of which has an asymptotic performance ratio of 3.25. Their first algorithm sorts the items by height and then divides them into groups of area (in the first two dimensions) at most 7/16, so that they can be packed into a single layer by Theorem 35.1. They improve on this by classifying items with similar bottoms, and packing similar items together into layers. Two items have similar bottoms if both their length and their width fall into the same class when classified by the HARMONIC algorithm. For the case where all items have square bottoms, the ratio improves to 2.6875.

Two years later, the same authors [48] presented an online algorithm with asymptotic performance ratio arbitrarily close to $h_\infty^2 \approx 2.89$ for three-dimensional strip packing. At the time, there was no better *offline* approximation known. This algorithm uses the HARMONIC algorithm as a subroutine in both horizontal dimensions (i.e., to find a strip for a two-dimensional item, and a place inside a strip for a one-dimensional item), and a geometric rounding for the heights. The paper actually discusses several online algorithms for this problem and only mentions the use of HARMONIC in the summary section. The authors [47] note that the improvement in the asymptotic performance ratio compared with the approximation algorithm from their earlier paper only comes at the cost of a high additive constant.

In 1997, Miyazawa and Wakabayashi [49] improved the offline upper bound to 2.66994 (2.36 for items with square bottoms). This algorithm places columns of similar items next to each other in the strip, thus avoiding the layer structure of the previous algorithms. The algorithm is quite involved and its description takes three pages. This remains the best result to date.

35.5.2 Rotations

In the case where rotations are allowed, it becomes relevant what exactly the dimensions of the strip are. In two-dimensional strip packing, this does not really play a part, but in column packing, the base of the column might not be a square. However, if the base is not a square but may be an arbitrary rectangle, then having the ability to rotate items horizontally (leaving the top side unchanged) does not help, as was shown by Miyazawa and Wakabayashi [4]. The idea is that in this case it is possible to scale the input so that the smallest width of an item is still larger than the length of the base of the strip, so that no item can be rotated and still fit inside the strip. For this reason, in this section we focus on the case where the base of the strip is a square.

Epstein and van Stee [6] give an approximation algorithm with asymptotic worst-case ratio of $9/4 = 2.25$, improving on the upper bound of 2.76 by Miyazawa and Wakabayashi [5]. The special case where only rotations that leave the top side of items at the top are allowed has received more attention. It was introduced by Li and Cheng [50] as a model for a job scheduling problem in partitionable mesh connected systems. Here each job i is given by a triple (x_i, y_i, t_i), meaning that job i needs a submesh of dimensions $x_i \times y_i$ or $y_i \times x_i$ for t_i time units. They give an algorithm for minimizing the makespan (i.e., the height of the packing), which has asymptotic performance bound $4\frac{4}{7}$. This was improved to 2.543 by Miyazawa and Wakabayashi [5] and finally to 2.25 by Epstein and van Stee [6].

35.6 Three- and More Dimensional Bin Packing

At present, the online-bounded space algorithm from Epstein and van Stee [20] is the best (online or offline) algorithm for packing multidimensional items into bins for any dimension $d \geq 3$. Clearly, this problem is APX-hard as well since it includes the two-dimensional bin packing problem as a special case [1].

Blitz et al. [30] gave a lower bound of 2.111 for online algorithms for $d = 3$. However, there is no good lower bound known for larger dimensions: nothing above 3. It appears likely that the asymptotic performance bound of any online algorithm must grow with the dimension.

For the special case of packing hypercubes online in dimensions $d \geq 4$, there is no better lower bound than the 4/3 given by Coppersmith and Raghavan [22] (which works in any dimension $d \geq 2$).

The bounded space algorithm by Epstein and van Stee [20] for this problem has a performance ratio that is sublinear in d: it is $O(d/\log d)$ and $\Omega(\log d)$.

For $d = 3$ (online cube packing), Miyazawa and Wakabayashi [51] showed that the algorithm of Coppersmith and Raghavan [22] has an asymptotic performance bound of 3.954. Epstein and van Stee [31] give an algorithm with asymptotic performance ratio at most 2.9421, and a lower bound of 1.6680.

Furthermore, the same authors [32] give bounds for the performance of their bounded space algorithm from Ref. [20], showing that its performance ratio lies between 2.95642 and 3.0672.

As was seen in Section 35.4.2, we can do even better offline. Before Bansal and Sviridenko [1] and Correa and Kenyon [3] gave their asymptotic polynomial-time approximation scheme for any dimension $d \geq 2$, Miyazawa and Wakabayashi [51] gave two approximation algorithms, of which the best had an asymptotic performance ratio of 2.6681. Soon afterwards, Kohayakawa et al. [37] presented their paper which we discussed in Section 35.4.2 as well. For $d = 3$, its asymptotic performance bound is $46/27 + \varepsilon \approx 1.7037\ldots + \varepsilon$.

35.7 Vector Packing

In this section we discuss the nongeometric version of multidimensional bin packing. The d-dimensional "vector packing" or "vector bin packing" problem is defined as follows. The bins are instances of the "all-1" vector $(1, 1, \ldots, 1)$ of length d. Items are d-dimensional vectors, whose components are all in $[0, 1]$. A packing is valid if the vector sum of all items assigned to one bin does not exceed the capacity of the bin (i.e., 1) in any component. Since all bins are identical, the goal is to minimize the number of bins used.

The problem can be seen as a scheduling problem with limited resources. The machines (with correspond to bins) have fixed capacities of several resources as memory, running time, and access to other computers. The items in this case are jobs that need to be run, each job requires a certain amount of each resource. Another application arises from viewing the problem as a storage allocation problem. Each bin has several qualities as volume and weight. Each item requires a certain amount of each quality. Both applications are relevant to both offline and online environments.

For many years there were very few results on this problem. In the first paper which obtained an APTAS for classical bin packing, Fernandez de la Vega and Lueker [19] imply a $(d + \varepsilon)$-approximation for the vector packing problem. This improved very slightly on some online results. These results were an upper bound of $d + 1$ on the performance ratio of any algorithm of which the output never has two bins that can be combined, given by Kou and Markowsky [52], and a tight bound on the performance of First Fit of $d + \frac{7}{10}$, given by Garey et al. [53]. Note that this is a generalization of the tight bound of $\frac{17}{10}$ for First Fit in one dimension.

Since these results were obtained, for a while there was hope that an APTAS would be found for this problem. However, Woeginger [2] proved that unless $P = NP$, there cannot be such an APTAS, already for two-dimensional vectors. Clearly, more restricted classes of vectors may still admit an APTAS. One such type of input is one where there is a total order on all vectors. In [54], Caprara et al. showed that an APTAS for this problem indeed exists.

The offline result for the general case was finally improved by Chekuri and Khanna [55]. They designed an algorithm of asymptotic performance $1 + \varepsilon d + O(\ln \frac{1}{\varepsilon})$. If d is seen as a constant, the best ratio achieved in this way is $O(\ln d)$. They proved that for an arbitrary d, it is APX-hard to approximate the problem within a factor of $d^{\frac{1}{2}-\varepsilon}$ for every fixed positive ε. This was shown using a reduction from graph coloring.

The online result was not improved since 1976. Lower bounds on the performance ratio of online algorithms, which tend to 2 as d grows, were shown by Galambos et al. [56]. Improved lower bounds were given by Blitz et al. [30], but this construction also tends to 2 as d grows.

As for the absolute approximation ratio, Kellerer and Kotov [57] designed an algorithm for two-dimensional vector packing with absolute approximation ratio of at most 2. Recently, Erlebach [58] showed a nonconstant lower bound on the absolute performance ratio for this problem. Interestingly, the method is similar to the one used by Chekuri and Khanna to show the hardness of approximation. The lower bound holds for the asymptotic performance ratio if d is not seen as a constant, that is, for arbitrary d.

As for variable-sized packing, the online problem was studied by Epstein [59]. In this problem, the algorithm may use bins out of a given finite subset. This subset contains the standard "all-1" vector, and possibly other vectors. The cost of a bin is the sum of its components. She showed that there exists a finite set where an online algorithm can achieve a performance of ratio of $1 + \varepsilon$ (by defining the class of bins to be dense enough), whereas for another set (which contains except for the "all-1" bin only bins that have relatively small components), the ratio must be linear in the dimension. Clearly, no matter what the set is, there exists a simple algorithm with linear performance ratio.

Analogous to the bin covering problem, we can define the vector covering problem, where the vector sum of all vectors assigned to one bin is *at least* 1 in every component. This problem was studied by Alon et al. [60]. In this paper it was shown that the performance ratio of any online algorithm is at least $d + \frac{1}{2}$. A linear upper bound of $2d$ is achieved by an algorithm that partitions the input into classes. The same paper contains offline results as well. An algorithm of performance guarantee $O(\log d)$ is presented as well as a simple and fast 2-approximation for $d = 2$. In Ref. [61] some results on variable-sized vector covering are given. These results focus on cases where all bins are vectors of zeros and ones. The benefit of a covered bin is the sum of its nonzero components. The considered cases for the bin sets are as follows: a set which consists of a single type of bin, a set of all unit vectors (all components are zero except for one), unit prefix vectors (some prefix of the vector consists of ones only), and the set of all zero–one vectors.

35.8 Variations

35.8.1 Rectangle Stretching

Imreh [62] studied an oriented online strip packing problem where rectangles can be stretched in a way that results in a larger height but the original area. Note that allowing stretching that increases the width makes the problem trivial as all items would be stretched to have the same width as the bin. He showed that the offline problem is polynomially solvable, and that if the online problem is considered under the asymptotic performance ratio measure (and assuming an upper bound of 1 on the original height of any rectangle), then the performance ratio can be made arbitrarily close to 1. Therefore, the main results are for the absolute performance ratio. There are algorithms of performance ratios 6 and 4, and a lower bound of 1.73 on the performance ratio of any online algorithm.

35.8.2 Items Appear from the Top

A "Tetris-like" online model was studied in a few papers. This is similar to strip packing, however, in this model, a rectangle cannot be placed directly in its designated area, but it arrives from the top as in the Tetris game, and should be moved continuously around only in the free space until it reaches its place (see Figure 35.4), and then cannot be moved again.

In [63], the model was introduced by Azar and Epstein. In that paper, both the rotatable and the oriented models were studied. For the rotatable model, a 4-approximation algorithm was designed. The situation for the oriented problem is more difficult, as no algorithm with constant approximation ratio exists for

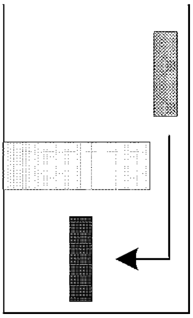


FIGURE 35.4 The process of packing an item in the "Tetris-like" model.

unrestricted inputs. If the width of all items is bounded below by ϵ and/or bounded above by $1 - \epsilon$, the authors showed a lower bound of $\Omega(\sqrt{\log \frac{1}{\epsilon}})$ on the performance ratio of any online algorithm for any deterministic or randomized algorithm. Restricting the width, they designed an $O(\log \frac{1}{\epsilon})$-approximation algorithm.

The oriented version of the problem was studied by Coffman et al. [64]. They assume a probabilistic model where item heights and widths are drawn from a uniform distribution on [0, 1]. They show that any online algorithm which packs n items has an asymptotic expected height of at least $0.313827n$ and design an algorithm of asymptotic expected height of $0.369764n$.

35.8.3 Dynamic Bin Packing

A multidimensional version of a dynamic bin packing model, which was introduced in Ref. [65] for the one-dimensional case, was studied recently by Epstein and Levy [66]. This is an online model where items do not only arrive but may also leave. Each event is an arrival or a departure of an item. Durations are not known in advance, that is, an algorithm is notified about the time that an item leaves only upon its departure. An algorithm may rearrange the locations inside bins, but the items may not migrate between bins. In Ref. [66], the same problem was studied in multiple dimensions.

In two dimensions, they designed a 4.25-approximation algorithm for dynamical packing of squares, and provided a lower bound of 2.2307 on the performance ratio. For rectangles the upper and lower bounds are 8.5754 and 3.7, respectively. For three-dimensional cubes they presented an algorithm which is a 5.37037-approximation, and a lower bound of 2.117. For three-dimensional boxes, they supplied a 35.346-approximation algorithm and a lower bound of 4.85383. For higher dimensions, they define and analyze the algorithm NFDH for the offline box packing problem. This algorithm was studied before for rectangle packing (two-dimensional only) [8], and for square and cube packing for any dimension [7,37], but not for box packing. For d-dimensional boxes they provided an upper bound of $2 \cdot 3.5^d$ and a lower bound of $d + 1$. Note that, as already mentioned in this survey, the best bound known for the regular offline multidimensional box packing problem is exponential as well. For d-dimensional cubes they provided an upper bound of $O\left(\frac{d}{\ln d}\right)$ and a lower bound of 2.

One earlier paper by Coffman and Gilbert [67] studies a related problem. In this problem, squares of a bounded size, which arrive and leave at various times, must be kept in a single bin. The paper gives lower bounds on the size of such a bin, so that all squares can fit. It is not allowed to rearrange the locations in the bin.

35.8.4 Packing Rectangles in a Single Rectangle

Another version is concerned with maximizing the number, area, or weight of a subset of the input rectangles that can be packed into a larger rectangle (of given height and width). The maximization problem with respect to the number of rectangles was studied already in 1983 by Baker et al. [68]. They designed an asymptotic $\frac{4}{3}$-approximation. This offline problem was recently studied by Jansen and Zhang [69,70]. The first paper considered the case of weighted rectangles and maximizing the total weight packed, whereas the second one considered unweighted rectangles and maximizing the number of packed rectangles. The problem is considered without rotation.

In [69], Jansen and Zhang proved that there exists an asymptotic FPTAS and an absolute PTAS, for packing squares into a rectangle. For rectangles they gave an approximation algorithm with asymptotic ratio of at most 2, and a simple one with an absolute ratio of $2 + \varepsilon$. In [70], Jansen and Zhang gave a more complicated algorithm for the weighted case with an absolute ratio of $2 + \varepsilon$. This algorithm has higher running time than the one for the unweighted problem. A special case of weights is simply the area of rectangles. The area maximization problem was studied by Caprara and Monaci [71]. They designed an algorithm with (absolute) approximation ratio of $3 + \varepsilon$.

An online version was studied by Han et al. [72]. In this version, we are given a unit square bin, rectangles arrive online, and the algorithm needs to decide whether to accept an arriving rectangle or not. The goal is again to maximize the packed area. They showed that if the algorithm is not allowed to remove rectangles accepted in the past, no algorithm with constant approximation ratio exists. This holds already for squares. It is easy to see that this holds with the following example. Take a first square which is very small and another one which fills the bin completely. An algorithm must accept the first square and therefore cannot accept the larger one. Next, they show that no algorithm with constant approximation ratio exists for rectangles, even if the algorithm is allowed to remove previously accepted rectangles. Therefore, the paper studies removable square packing. Before describing the results, we discuss a related paper that was used in this paper.

Januszewski and Lassak [73] studied a similar problem from the point of view of finding a threshold $\alpha \leq 1$ such that a set of squares of total area of at most α can be always packed online in a bin, without rearranging the contents of the bin. They showed that $\frac{5}{16}$ is lower bound on α. Moreover, they considered this problem for multidimensional cubes, and showed a lower bound of $\frac{1}{2^d-1}$ for $d \geq 5$. For the packing they used a nice tool which they called bricks. A brick is a rectangle, where the ratio of the maximum between height and width to the minimum between the two remains the same after cutting the rectangle into two identical parts. Clearly, this can work if the ratio is $\sqrt{2}$.

Han et al. [72] adopted this method. They showed that any algorithm has performance ratio of at least $\phi + 1 \approx 2.618$. They designed a matching algorithm for the case where rearranging is allowed, and a 3-approximation algorithm without rearranging. A direct consequence is that a lower bound on α for two dimensions is $\frac{1}{3}$.

Finally, another related problem is packing squares or rectangles into a square or rectangle of minimum size, where arbitrary rotations are allowed (not just over 90°). For example, five unit squares, can be packed inside a square with side $2 + \frac{1}{2}\sqrt{2}$, by placing four squares in the corners and one in the center at a 45° angle. For a survey on packing equal squares into a square; see for example, Ref. [74]. Novotný [75] showed that any set of squares with total area 1 can be packed in a rectangle of area at most 1.53 (without rotations).

References

[1] Bansal, N. and Sviridenko, M., New approximability and inapproximability results for 2-dimensional packing, *Proc. of SODA,* ACM/SIAM, 2004, p. 189.

[2] Woeginger, G. J., There is no asymptotic PTAS for two-dimensional vector packing, *Inf. Process. Lett.*, 64(6), 293, 1997.

[3] Correa J. R. and Kenyon, C., Approximation schemes for multidimensional packing, *Proc. of SODA,* ACM/SIAM, 2004, p. 179.

[4] Miyazawa, F. K. and Wakabayashi, Y., Approximation algorithms for the orthogonal z-oriented 3-d packing problem, *SIAM J. Comput.*, 29(3), 1008, 1999.

[5] Miyazawa, F. K. and Wakabayashi, Y., Packing problems with orthogonal rotations, in *Theoretical Informatics, 6th Latin American Symp.*, Martin Farach-Colton, Ed., Lecture Notes in Computer Science, Vol. 2976, Springer, Berlin, 2004, p. 359.

[6] Epstein, L. and van Stee, R., This side up! *ACM Transactions on Algorithms*, 2(2), 228, 2006.

[7] Meir, A. and Moser, L., On packing of squares and cubes, *J. Combin. Theor.*, 5, 126, 1968.

[8] Coffman, E. G., Jr., Garey, M. R., Johnson, D. S., and Tarjan, R. E., Performance bounds for level oriented two-dimensional packing algorithms, *SIAM J. Comput.*, 9, 808, 1980.

[9] Kenyon, C. and Rémila, E., A near optimal solution to a two-dimensional cutting stock problem, *Math. Oper. Res.*, 25(4), 645, 2000.

[10] Baker, B. S., and Schwartz, J. S., Shelf algorithms for two-dimensional packing problems, *SIAM J. Comput.*, 12, 508, 1983.

[11] Csirik, J. and Woeginger, G. J., Shelf algorithms for on-line strip packing, *Inf. Process. Lett.*, 63, 171, 1997.

[12] van Vliet, A., An improved lower bound for online bin packing algorithms, *Inf. Process. Lett.*, 43, 277, 1992.

[13] Baker, B. S., Brown, D. J., and Katseff, H. P., A 5/4 algorithm for two-dimensional packing, *J. Algorithms*, 2, 348, 1981.

[14] Brown, D. J., Baker, B. S., and Katseff, H. P., Lower bounds for on-line two-dimensional packing algorithms, *Acta Informatica*, 18, 207, 1982.

[15] Baker, B. S., Coffman Jr., E. G., and Rivest, R. L., Orthogonal packings in two dimensions, *SIAM J. Comput.*, 9, 846, 1980.

[16] Schiermeyer, I., Reverse-fit: a 2-optimal algorithm for packing rectangles, *Proc. of ESA,* 1994, p. 290.

[17] Steinberg, A., A strip-packing algorithm with absolute performance bound 2, *SIAM J. Comput.*, 26(2), 401, 1997.

[18] Sleator, D. K., A 2.5 times optimal algorithm for packing in two dimensions, *Inf. Process. Lett.*, 10, 37, 1980.

[19] Fernandez de la Vega, W. and Lueker, G. S., Bin packing can be solved within $1 + \varepsilon$ in linear time, *Combinatorica*, 1, 349, 1981.

[20] Epstein, L. and van Stee, R., Optimal online algorithms for multidimensional packing problems, *SIAM J. Comput.*, 35(2), 431, 2005.

[21] Jansen, K., and van Stee, R., On strip packing with rotations, *Proc. of STOC,* ACM, 2005, p. 755.

[22] Coppersmith, D. and Raghavan, P., Multidimensional online bin packing: algorithms and worst-case analysis, *Oper. Res. Lett.*, 8, 17, 1989.

[23] Csirik, J., Frenk, J. B. G., and Labbe, M., Two dimensional rectangle packing: on line methods and results, *Discrete Appl. Math.*, 45, 197, 1993.

[24] Csirik, J. and van Vliet, A., An online algorithm for multidimensional bin packing, *Oper. Res. Lett.*, 13, 149, 1993.

[25] Li, K. and Cheng, K.-H., A Generalized Harmonic Algorithm for On-Line Multi-Dimensional Bin Packing, Technical report UH-CS-90-2, University of Houston, January 1990.

[26] Seiden, S. S. and van Stee, R., New bounds for multi-dimensional packing, *Algorithmica*, 36(3), 261, 2003.

[27] Galambos, G., A 1.6 lower bound for the two-dimensional online rectangle bin packing, *Acta Cybern.*, 10, 21, 1991.

[28] Galambos, G., and van Vliet, A., Lower bounds for 1-, 2-, and 3-dimensional online bin packing algorithms, *Computing*, 52, 281, 1994.

[29] van Vliet, A., Lower and Upper Bounds for Online Bin Packing and Scheduling Heuristics, Ph.D. thesis, Erasmus University, Rotterdam, The Netherlands, 1995.

[30] Blitz, D., van Vliet, A., and Woeginger, G. J., Lower bounds on the asymptotic worst-case ratio of online bin packing algorithms, unpublished manuscript, 1996.

[31] Epstein, L. and van Stee, R., Online square and cube packing, *Acta Informatica*, 41(9), 595, 2005.

[32] Epstein, L. and van Stee, R., Bounds for Online Bounded Space Hypercube Packing, Technical report SEN-E0417, CWI, Amsterdam, 2004.

[33] Chung, F. R. K., Garey, M. R., and Johnson, D. S., On packing two-dimensional bins, *SIAM J. Algebraic Discrete Methods*, 3, 66, 1982.

[34] Caprara, A., Packing 2-dimensional bins harmony, *Proc. of FOCS,* 2002, p. 490.

[35] Leung, J. Y.-T., Tam, T. W., Wong, C. S., Young, G. H., and Chin, F. Y. L., Packing squares into a square, *J. Parallel Distributed Comput.*, 10, 271, 1990.

[36] Ferreira, C. E., Miyazawa, F. K., and Wakabayashi, Y., Packing squares into squares, *Pesquisa Operacional*, 19(2), 223, 1999.

[37] Kohayakawa, Y., Miyazawa, F. K., Raghavan, P., and Wakabayashi, Y., Multidimensional cube packing, *Algorithmica*, 40(3), 173, 2004.

[38] Epstein, L. and van Stee, R., Optimal online bounded space multidimensional packing, *Proc. of SODA,* ACM/SIAM, 2004, p. 207.

[39] Bansal, N., Lodi, A., and Sviridenko, M., A tale of two dimensional bin packing, *Proc of FOCS,* 2005.

[40] Caprara, A., Lodi, A., and Monaci, M., Fast approximation schemes for the two-stage, two-dimensional bin packing problem, *Math. Oper. Res.*, 30, 150, 2005.

[41] Zhang, G., A 3-approximation algorithm for two-dimensional bin packing, *Oper. Res. Lett.*, 33(2), 121, 2005.

[42] van Stee, R., An approximation algorithm for square packing, *Oper. Res. Lett.*, 32(6), 535, 2004.

[43] Bansal, N. and Sviridenko, M., Two-dimensional bin packing with one dimensional resource augmentation, manuscript, 2005.

[44] Epstein, L., Two dimensional packing: the power of rotation, *Proc. 28th Int. Symp. Mathematical Foundations of Computer Science (MFCS'2003)*, 2003, p. 398.

[45] Fujita, S. and Hada, T., Two-dimensional on-line bin packing problem with rotatable items, *Theor. Comput. Sci.*, 289(2), 939, 2002.

[46] Correa, J. R., Resource augmentation in two-dimensional packing with orthogonal rotations, *Oper. Res. Lett.*, 34(1), 85, 2006.

[47] Li, K. and Cheng, K.-H., On three-dimensional packing, *SIAM J. Comput.*, 19(5), 847, 1990.

[48] Li, K. and Cheng, K.-H., Heuristic algorithms for online packing in three dimensions, *J. Algorithms*, 13, 589, 1992.

[49] Miyazawa, F. K. and Wakabayashi, Y., An algorithm for the three-dimensional packing problem with asymptotic performance analysis, *Algorithmica*, 18(1), 122, 1997.

[50] Li, K. and Cheng, K.-H., Generalized First-Fit algorithms in two and three dimensions, *Int. J. Found. Comput. Sci.*, 2, 131, 1990.

[51] Miyazawa, F. K. and Wakabayashi, Y., Cube packing, *Theor. Comput. Sci.*, 297(1–3), 355, 2003.

[52] Kou, L. T. and Markowsky, G., Multidimensional bin packing algorithms, *IBM J. Res. Dev.*, 21, 443, 1977.

[53] Garey, M. R., Graham, R. L., Johnson, D. S., and Yao, A. C., Resource constrained scheduling as generalized bin packing, *J. Combin. Theor. (Series A)*, 21, 257, 1976.

[54] Caprara, A., Kellerer, H., and Pferschy, U., Approximation schemes for ordered vector packing problems, *Naval Res. Logistics*, 92, 58, 2003.

[55] Chekuri, C. and Khanna, S., On multidimensional packing problems, *SIAM J. Comput.*, 33(4), 837, 2004.

[56] Galambos, G., Kellerer, H., and Woeginger, G. J., A lower bound for online vector packing algorithms, *Acta Cybern.*, 10, 23, 1994.

[57] Kellerer, H. and Kotov, V., An approximation algorithm with absolute worst-case performance ratio 2 for two-dimensional vector packing, *Oper. Res. Lett.*, 31(1), 35, 2003.

[58] Erlebach, T., Private communication, 2005.

[59] Epstein, L., On variable sized vector packing, *Acta Cybern.*, 16, 47, 2003.

[60] Alon, N., Azar, Y., Csirik, J., Epstein, L., Sevastianov, S. V., Vestjens, A., and Woeginger, G. J., On-line and off-line approximation algorithms for vector covering problems, *Algorithmica*, 21, 104, 1998.

[61] Epstein, L., On-line variable sized covering, *Inf. Comput.*, 171(2), 294, 2001.

[62] Imreh, C., Online strip packing with modifiable boxes, *Oper. Res. Lett.*, 66, 79, 2001.

[63] Azar, Y. and Epstein, L., On two dimensional packing, *J. Algorithms*, 25(2), 290, 1997. Also in *Proc. of SWAT'96*, p. 321.

[64] Coffman, E. G., Jr., Downey, P. J., and Winkler, P. M., Packing rectangles in a strip, *Acta Inf.*, 38(10), 673, 2002.

[65] Coffman, E. G., Jr., Garey, M. R., and Johnson, D. S., Dynamic bin packing, *SIAM J. Comput.*, 12, 227, 1983.

[66] Epstein L. and Levy, M., Dynamic multi-dimensional bin packing, manuscript, 2005.

[67] Coffman Jr., E. G. and Gilbert, E. N., Dynamic, first-fit packings in two or more dimensions, *Inf. Control*, 61(1), 1, 1984.

[68] Baker, B. S., Calderbank, A. R., Coffman, E. G., Jr., and Lagarias, J. C., Approximation algorithms for maximizing the number of squares packed into a rectangle, *SIAM J. Algebraic Discrete Methods*, 4(3), 383, 1983.

[69] Jansen, K. and Zhang, G., On rectangle packing: Maximizing benefits, *Proc. of SODA*, 2004, p. 204.

[70] Jansen, K. and Zhang, G., Maximizing the number of packed rectangles, *Proc. 9th Scandinavian Workshop on Algorithm Theory (SWAT'04)*, 2004, p. 362.

[71] Caprara, A. and Monaci, M., On the 2-dimensional knapsack problem, *Oper. Res. Lett.*, 32, 5, 2004.

[72] Han, X., Iwama, K., and Zhang, G., Online removable square packing, *Proc. 3rd Workshop on Approximation and Online Algorithms (WAOA 2005)*, Springer, Berlin, 2005, p. 216.

[73] Januszewski, J. and Lassak, M., Online packing sequences of cubes in the unit cube, *Geometriae Dedicata*, 67, 285, 1997.

[74] Friedman, E., Packing unit squares in squares: a survey and new results, *Electron. J. Combin.*, 7, 2005.

[75] Novotný, P., On packing of squares into a rectangle, *Arch. Math.*, 32, 75, 1996.

36

Practical Algorithms for Two-Dimensional Packing

Shinji Imahori
University of Tokyo

Mutsunori Yagiura
Nagoya University

Hiroshi Nagamochi
Kyoto University

36.1 Introduction

Cutting and packing problems consist of placing a given set of (small) items into one or more (larger) objects without overlap so as to minimize/maximize a given objective function. This is a combinatorial optimization problem with many important applications in the wood, glass, steel, and leather industries, as well as in Large Scale Integration (LSI) and Very Large Scale Integration (VLSI) design, newspaper paging, and container and truck loading. For several decades, cutting and packing has attracted the attention of researchers in various areas including operations research, computer science, manufacturing, etc.

Cutting and packing problems can be classified using different criteria. The dimensionality of a problem is one of such criteria, and most problems are defined over one, two, or three dimensions. In this chapter we consider two-dimensional problems. The next criterion to classify two-dimensional packing problems is the shape of items to pack. We focus on the *rectangle packing problem* in this chapter. This problem has been actively studied in the past few decades. When the shapes of the items to be packed are polygons or arbitrary shapes, the problem is called *irregular packing*. We also discuss in this chapter recent research in this area.

Almost all two-dimensional packing problems are known to be NP-hard, and hence it is impossible to solve them exactly in polynomial time unless P = NP. Therefore, heuristics and metaheuristics are very important to design practical algorithms for these problems. We survey practical algorithms for two-dimensional packing problems in this chapter. We also survey various schemes used to represent solutions to rectangle packing problem, and introduce algorithms based on these *coding schemes*.

The remainder of this chapter is organized as follows: Section 36.2 defines the rectangle packing problem and its variations. Section 36.3 introduces coding schemes for the rectangle packing problem, which are used to represent solutions. Section 36.4 presents heuristic algorithms, from the traditional to the latest ones, for the rectangle packing problem. Section 36.5 discusses practical algorithms based on metaheuristics. Sections 36.4 and 36.5 discuss computational results for the various algorithms on benchmark instances. Section 36.6 defines the irregular packing problem, and practical algorithms for this problem are presented.

36.2 Rectangle Packing Problem

We consider the following two-dimensional rectangle packing problem. We are given n items (small rectangles) $I = \{1, 2, \ldots, n\}$, each of which has width w_i and height h_i, and one or many large objects (rectangles). We are required to place the items orthogonally without any overlap (an edge of each item is parallel to an edge of the object) so as to minimize (or maximize) a given objective function. The rectangle packing problem arises in many industrial applications, often with slightly different constraints, and many variants of this problem have been considered in the literature. The following characteristics are important to classify the problems [1,2]: type of assignment, assortment of objects, and assortment of items. We will review some specific variations of the rectangle packing problem in this section. We should mention two more important constraints for the rectangle packing problem: orientation and guillotine cut constraint. As for the orientation of the items, we usually assume that "each rectangle has a given fixed orientation" or "each rectangle can be rotated by 90°." Rotation of items is not allowed in newspaper paging or when the items to be cut are decorated or corrugated, whereas orientation is free in the case of plain materials, and so on. Guillotine cut constraint signifies that the items must be obtained through a sequence of edge-to-edge cuts parallel to the edges of the large object (see Figure 36.1 for an example), which is usually imposed by technical limitations of the automated cutting machines or the material.

We introduce six types of rectangle packing problems that have been actively studied. For simplicity, we define the problems assuming that each item has a fixed orientation and the guillotine cut constraint is not imposed unless otherwise stated. It is straightforward to extend our definitions for other cases where each item can be rotated by 90° and/or the guillotine cut constraint is imposed. We first consider two types of typical rectangle packing problems with one large rectangular object, which may grow in one or two dimensions, where all the items are placed disjointly. The problems are called *strip packing* and *area minimization.*

Strip packing problem. We are given n items (small rectangles) each having width w_i and height h_i, and one large object (called a strip) whose width W is fixed, but its height H is variable. The objective is to minimize the height H of the strip such that all items can be packed into the strip.

Area minimization problem. We are given n items each having width w_i and height h_i, and one large rectangular object, where both its width W and height H are variables. The objective is to minimize the area WH of the object such that all items can be packed into the object.

In Sections 36.3–36.5, we focus mainly on the strip packing (and area minimization) problem, which is formally formulated as the following mathematical program:

$$\begin{array}{lll} \text{minimize} & \text{the height of the strip}\, H & \text{(or the area of the large object } WH) \\ \text{subject to} & 0 \le x_i \le W - w_i, & \text{for all } i \in I \qquad (36.1) \\ & 0 \le y_i \le H - h_i, & \text{for all } i \in I \qquad (36.2) \end{array}$$

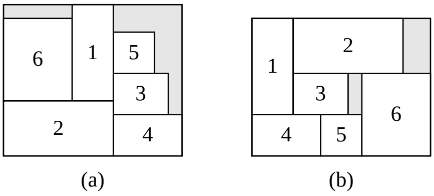


FIGURE 36.1 Examples of placements with/without guillotine cut constraint. (a) Guillotine cut; (b) nonguillotine.

At least one of the next four inequalities holds for every pair i and j:

$$x_i + w_i \leq x_j \tag{36.3}$$

$$x_j + w_j \leq x_i \tag{36.4}$$

$$y_i + h_i \leq y_j \tag{36.5}$$

$$y_j + h_j \leq y_i \tag{36.6}$$

where (x_i, y_i) is the coordinate of the lower left corner of an item i. Constraints (36.1) and (36.2) mean that all items must be placed into the large object, and Constraints (36.3)–(36.6) mean that no two items overlap (i.e., each inequality signifies one of the four relative locations: left-of, right-of, above, and below).

Two other rectangle packing problems are the *two-dimensional bin packing* and *knapsack* problems that have (many or one) fixed-sized objects.

Two-dimensional bin packing problem. We are given a set of items, where each item i has width w_i and height h_i, and an unlimited number of large objects (rectangular bins) having identical width W and height H. The objective is to minimize the number of rectangular bins used to place all the items.

Two-dimensional knapsack problem. We are given a set I of items, where each item $i \in I$ has width w_i, height h_i, and value c_i. We are also given a rectangular knapsack with fixed width W and height H. The objective is to find a subset $I' \subseteq I$ of items with the maximum total value $\sum_{i \in I'} c_i$ such that all items $i \in I'$ can be packed into the knapsack.

For the two-dimensional bin packing problem, Lodi et al. [3] proposed practical heuristic and metaheutistic algorithms and performed computational experiments on various benchmark instances. For the two-dimensional knapsack problem, Wu et al. [4] proposed heuristic algorithms that are effective for many test instances.

We should also mention the following two problems: *two-dimensional cutting stock* and *pallet loading.* For some industrial applications, such as mass production manufacturing, many small items of an identical shape or relatively few classes of shapes are packed into the objects. The following two problems are useful for modeling these situations.

Two-dimensional cutting stock problem. We are given a set of items each with width w_i, height h_i, and demand d_i. We are also given an unlimited number of objects having identical width W and height H. The objective is to minimize the number of objects used to place all the items (i.e., for each i, we place d_i copies of item i into the objects).

Pallet loading problem. We are given sufficiently large number of items with identical size (w, h), and one large rectangular object with size (W, H). The objective is to place the maximum number of items into the object, where each item can be rotated by 90°.

Note that the pallet loading problem with a fixed orientation of items is trivial to solve. Among many studies on the two-dimensional cutting stock problem, Gilmore and Gomory [5] provided one of the earliest solution methods. They proposed a column generation scheme in which new cutting patterns are produced by solving a generalized knapsack problem. Recently, some practical algorithms for the two-dimensional cutting stock problem have been proposed [6,7]. Morabito and Morales [8] proposed a simple but effective algorithm for the pallet loading problem.

The complexity of the pallet loading problem is open (this problem is not known to be in class NP, because of the compact input description), whereas the other problems we defined in this section are known to be NP-hard.

36.3 Coding Schemes for Rectangle Packing

In this section, we review coding schemes for rectangle packing problems. For simplicity, we focus on the strip packing problem. An effective search will be difficult if we search the x and y coordinates of each item directly, since the number of solutions is uncountable and eliminating the overlap between items is not easy. To overcome this difficulty, various coding schemes have been proposed and many algorithms for rectangle packing problems are based on some coding schemes.

A *coding scheme* consists of a set of coded solutions, a mapping from coded solutions to placements, and a *decoding algorithm* that computes for a given coded solution the corresponding placement using the mapping. (The mapping is sometimes defined by the decoding algorithm.) Properties of a coding scheme (and a decoding algorithm) are given below:

1. There exists a coded solution that corresponds to an optimal placement.
2. The number of all possible coded solutions is finite, where a small total number is preferable provided that property 1 is satisfied.
3. Every coded solution corresponds to a feasible placement.
4. Decoding is possible in polynomial time. Fast algorithms are more desirable.

Some of the coding schemes in the literature satisfy all of the above four properties, but others do not.

One of the most popular coding schemes is to represent a solution by a permutation of the n items, where a coded solution (i.e., a permutation of n items) specifies the placement order. The number of all possible coded solutions is $O(n!)$, which is smaller than other coding schemes in the literature, and every permutation corresponds to a placement without items overlapping. A decoding algorithm computes a placement from a given coded solution by specifying the locations of the items one by one, which defines the mapping from coded solutions to placements and is sometimes called a *placement rule*. The decoding time complexity and the existence of a coded solution that corresponds to an optimal placement depend on the decoding algorithms. In the literature, a number of decoding algorithms for permutation coding schemes have been proposed. We will explain some typical decoding algorithms in Section 36.4, and compare them theoretically and experimentally.

We now explain different types of coding schemes for rectangle packing problems. The schemes we explain hereafter specify the relative locations for each pair of items by a coded solution. In other words, for every pair of items, a coded solution determines one of the four inequalities (36.3)–(36.6), which must be satisfied to avoid item overlap. The placement corresponding to a coded solution is the best one among those that satisfy the relative locations specified by the coded solution.

One of the most popular coding schemes of this type is to represent a solution by an n-leaf binary tree [9]. This coding scheme can represent only slicing structures (in other words, each placement obtained by this representation always satisfies the guillotine cut constraint). The leaves of a binary tree correspond to items, and each internal node has a label "h" or "v," where h stands for horizontal and v for vertical. This coding scheme uses $O(n)$ space to represent a solution, and the number of all possible coded solutions is $O(n!2^{5n-3}/n^{1.5})$ [9]. In this scheme, one of the four relative locations is assigned for each pair i and j of items as follows: If i is a left descendant of an internal node u with "h" label and j is a right descendant of the same internal node u (i.e., u is the least common ancestor of i and j), then we must place i to the left of j (i.e., $x_i + w_i \leq x_j$). If the label of the least common ancestor is "v," then we place i below j (i.e., $y_i + h_i \leq y_j$). Figure 36.2 shows an example of a binary tree representation and a placement that satisfies these constraints. For example, in the figure, there is an internal node with "h" label for which the node for item 6 is a left descendant and the node for item 1 is a right descendant, and hence item 6 is placed to the left of item 1; the node for item 4 is a left descendant and the node for item 3 is a right descendant of the least common ancestor with "v" label, and hence item 4 is placed below item 3 and so forth. For a given binary tree τ, let Π_τ denote the set of all placements that satisfy the above horizontal–vertical constraints. The placement corresponding to a coded solution τ is one of the best (i.e., the most compact) placements in Π_τ. Though Π_τ contains infinitely many placements for any τ, natural decoding algorithms

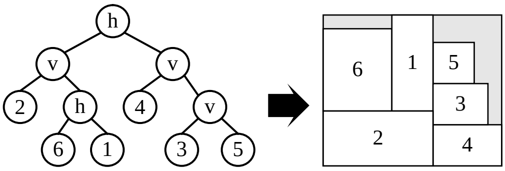


FIGURE 36.2 A binary tree representation τ and a solution $\pi \in \Pi_\tau$.

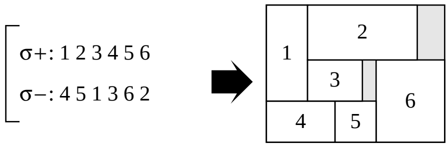


FIGURE 36.3 A sequence pair representation σ and a solution $\pi \in \Pi_\sigma$.

for this coding scheme run in linear time with respect to the number of items, and computes one of the best placements among Π_τ. Moreover, for any placement π that satisfy the guillotine cut constraint, there exists a binary tree τ that satisfies $\pi \in \Pi_\tau$. That is, the binary tree coding scheme satisfies all of the four desirable properties of a coding scheme if the guillotine cut constraint is imposed.

Murata et al. [10] proposed a coding scheme called *sequence pair*. For the sequence pair representation, a solution is represented by a pair of permutations $\sigma = (\sigma_+, \sigma_-)$ of the n items (see Figure 36.3 for an example). Based on this coded solution, we assign relative locations for each pair of items i and j as follows: If item i is before item j in both permutations σ_+ and σ_-, then item i must be placed to the left of j. If i is before j in σ_+ and after j in σ_-, then we place i above j. For example, in Figure 36.3, element 1 is before element 2 in both permutations, and hence item 1 is placed to the left of item 2; element 2 is before element 3 in permutation σ_+ and after element 3 in σ_-, and hence item 2 is placed above item 3 and so on. For a given pair of permutations $\sigma = (\sigma_+, \sigma_-)$, let Π_σ be the set of placements that satisfy the above constraints. The placement corresponding to a coded solution σ is one of the best placements in Π_σ. Murata et al. [10] proposed an $O(n^2)$ time decoding algorithm to obtain one of the best placements $\pi \in \Pi_\sigma$ for a given coded solution σ. Takahashi [11] improved the time complexity of the decoding algorithm to $O(n \log n)$; Tang et al. [12] further improved it to $O(n \log\log n)$. Moreover, for any feasible placement π, there exists a coded solution σ that satisfies $\pi \in \Pi_\sigma$ (such a pair of permutations σ can be computed in $O(n \log n)$ time [13]). That is, the sequence pair coding scheme satisfies all of the four desirable properties of a coding scheme.

Nakatake et al. [14] proposed a coding scheme called *bounded sliceline grid* (BSG). BSG consists of a set of small rooms that are separated by horizontal and vertical segments, where the number of rooms in the horizontal and vertical directions, denoted p and q, respectively, are parameters that satisfy $pq \geq n$ (see Figure 36.4[a] with $p = q = 6$). It introduces one of the four orthogonal relations (left-of, right-of, above, and below) uniquely for each pair of rooms (see Figure 36.4[b] and Figure 36.4[c]). In these figures,

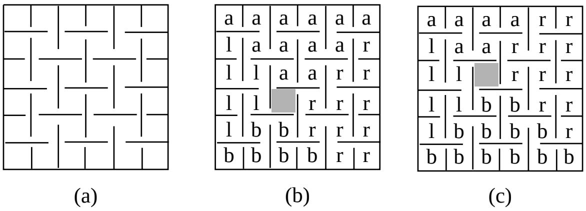


FIGURE 36.4 Rooms of bounded sliceline grid representation and relative locations.

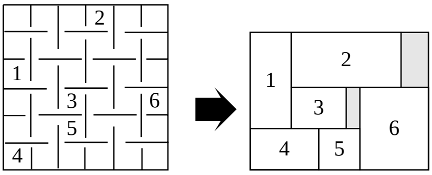


FIGURE 36.5 A bounded sliceline grid representation α and a solution $\pi \in \Pi_\alpha$.

a room with label l (resp., r, a, b) is left-of (resp., right-of, above, below) the shaded room. A solution is represented by an assignment of the items to rooms, where at most one item can be assigned to each room. The assigned items inherit the relations defined on the rooms. Figure 36.5 shows an example of an assignment of items to rooms and a placement that satisfy all specified constraints of relative locations. For example, in the figure, item 3 is placed to the right of item 1, item 3 is placed below item 2, item 3 is placed above item 4, and so forth. For a given assignment α, let Π_α be the set of all placements that satisfy the constraints given by α. The placement corresponding to a coded solution α is one of the best placements in Π_α. A decoding algorithm proposed by Nakatake et al. [14] runs in linear time with respect to the number of small rooms pq, and can find one of the best placements $\pi \in \Pi_\alpha$ for a given coded solution α. As for the existence of a coded solution that corresponds to an optimal placement, it is known that an assignment α such as $\pi \in \Pi_\alpha$ always exists for any placement π if and only if $p \geq n$ and $q \geq n$ hold. The BSG coding scheme with parameters $p \geq n$ and $q \geq n$ satisfies all of the four desirable properties of a coding scheme.

There are many other coding schemes that describe the relative locations for each pair of items. Guo et al. [15] proposed a tree representation called O-tree: Two ordered trees for the horizontal and vertical directions are used to represent a coded solution. This coding scheme can represent nonslicing structures and the number of all possible coded solutions is $O(n!2^{2n-2}/n^{1.5})$; this is smaller than the number of all coded solutions by the binary tree representation for slicing structures. There exists a coded solution corresponding to any placement π that satisfies the bottom left (BL) stability; that is, in the resulting placement, all items cannot be moved any further to the bottom or to the left. Chang et al. [16] extended the result of Guo et al. [15]. They proposed another tree representation called B*-tree; it is easy to implement this data structure and a decoding algorithm for B*-tree runs in linear time with respect to the number of items. Sakanushi et al. [17] proposed another coding scheme called quarter-state sequence: They utilized a string of items and labels to represent a solution and their decoding algorithm runs in linear time of the number of items.

36.4 Heuristics for Rectangle Packing

In this section, we describe heuristic algorithms for rectangle packing problems. We first explain some heuristic algorithms based on the permutation coding scheme. Those algorithms consist of two phases: (1) construct a permutation and (2) place the items one by one according to the permutation.

For the first phase, a standard strategy to construct a permutation is "a larger item has higher priority than a smaller one." To realize this, the items are sorted by some criteria, for example, decreasing height, decreasing width, or decreasing area. It is difficult to decide a priori which criterion is the best for numerous instances that arise in practice. Hence, many algorithms generate several permutations with different criteria, and apply a decoding algorithm to all such permutations.

Let us consider the second phase, that is, decoding algorithm for permutations. We first explain *level algorithms* in which the placement is obtained by placing items from left to right in rows forming levels (see Figure 36.6 for an example). The first level is the bottom of the object, and each subsequent level is along the horizontal line coinciding with the top of the tallest item packed on the level below.

The most popular level algorithms are the *next fit*, *first fit*, and *best fit* strategies, which are extended from the algorithms for the (one-dimensional) bin packing problem. Let i ($i = 1, 2, \ldots, n$) denote the

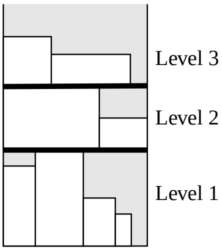


FIGURE 36.6 An example of level packing.

current item to be placed, and s be the level created most recently, where the bottom of the object is level 1 created at the beginning of an algorithm.

- *Next fit* strategy. Item i is packed on level s left justified (i.e., place it at the leftmost feasible position) if it fits. Otherwise, a new level ($s := s + 1$) is created and i is packed on it left justified.
- *First fit* strategy. We check whether or not item i fits from level 1 to level s, and pack it left justified on the first level where it fits. If no level can accommodate i, it is placed on a new level as in the next fit strategy.
- *Best fit* strategy. Item i is packed left justified on the level that minimizes the unused horizontal space among those where it fits. If no level can accommodate i, it is placed on a new level as in the next fit strategy.

Computation time of these algorithms is $O(n)$, $O(n \log n)$, and $O(n \log n)$, respectively, if appropriately implemented. The above strategies are illustrated by examples in Figure 36.7 (in this figure, items are sorted by decreasing height and are numbered accordingly). The resulting placements of these algorithms always satisfy the guillotine cut constraint. More precisely, they are the so-called two-stage guillotine placements in that they can be cut out in two stages: the first stage for horizontal cuts and the second stage for vertical cuts.

A different classical approach, and the most documented one, is the BL approach. The first algorithm of this type was proposed by Baker et al. [18] in 1980, and some variants of this method have been proposed in the last couple of decades. A common characteristic of this type of algorithms is to place items one by one at the BL stable positions; that is, in the resulting placement, all items cannot be moved any further to the bottom or to the left.

Baker et al. [18] used a BL rule that places each item at the leftmost point among the lowest possible positions. This approach is called *bottom left fill* (BLF) strategy in Refs. [19,20], which is illustrated by an example in Figure 36.8(a). The "x" marks in the figure show the BL stable positions. There are natural algorithms that require $O(n^3)$ time in the worst case for this strategy, and Hopper and Turton implemented one of them in their article [20]. Chazelle [21] devised an efficient algorithm that requires $O(n^2)$ time and $O(n)$ space in the worst case.

Jakobs [22] utilized another BL method: For each item, first place it at the top right location of the object and make successive sliding moves down and to the left alternately as far as possible (see an example

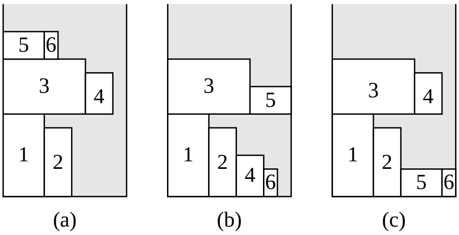


FIGURE 36.7 Three level algorithms for the strip packing problem: (a) next fit; (b) first fit; (c) best fit.

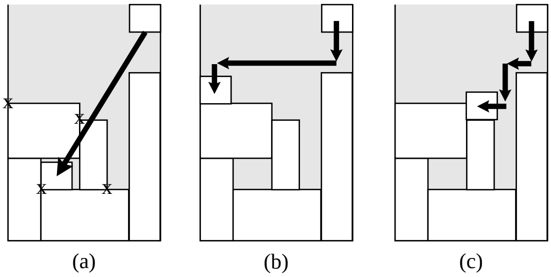


FIGURE 36.8 Three bottom left algorithms for the strip packing problem: (a) Baker et al.; (b) Jakobs; (c) Liu and Teng.

in Figure 36.8[b]). This strategy is called BL in Refs. [19,20] and it runs in $O(n^2)$ time, if appropriately implemented. We will see a comparison of BL and BLF algorithms through our computational experiments later on. Liu and Teng [23] developed another BL heuristics similar to Jakobs's algorithm. In their strategy, the downward movement has priority such that items slide leftward only if no downward movement is possible (see Figure 36.8[c]). This algorithm also runs in $O(n^2)$ time.

There are more algorithms which utilize the permutation to represent a solution. For example, Lodi et al. [3] proposed several decoding algorithms such as *floor ceiling, alternate directions,* and *touching perimeter,* and experimentally compared these algorithms with other decoding algorithms in the literature. Wu et al. [4] proposed complicated decoding algorithms, which runs in $O(n^4 \log n)$ or $O(n^5 \log n)$ time and has achieved certain computational success.

We discuss a different type of heuristic algorithm proposed by Burke et al. [19] in 2004. This method does not have a permutation of items to place, but dynamically decide the next item to place during the packing stage. More precisely, it finds the lowest available gap (LAG) within the large object and then places the item that best fits there (see Figure 36.9 for an example). This enables the algorithm to make informed decisions about which item should be packed next and where it should be placed. A natural implementation of this strategy runs in $O(n^2)$ time. We will also present experimental results for this algorithm.

At the end of this section, we compare typical heuristic algorithms through computational experiments. Test instances given by Hopper and Turton [20] were used for the experiments. There are seven different categories C1, C2, ..., C7 with the number of items ranging from 17 to 197, with each category having three instances. The optimal solution for all instances are known; these instances have placements without any dead space (in other words, these instances have perfect packings). The results of algorithms BL-R, BL-DW, BL-DH, BLF-R, BLF-DW, and BLF-DH are taken from Ref. [20], where BL (resp., BLF) means that items are placed with BL (resp., BLF) strategy into the object, and R, DW, and DH signify the types of permutations: R means random permutation and DW (resp., DH) means that items are sorted by decreasing width (resp., decreasing height). The results by Burke et al. reported in Ref. [19] algorithm (denoted BKW) are also shown.

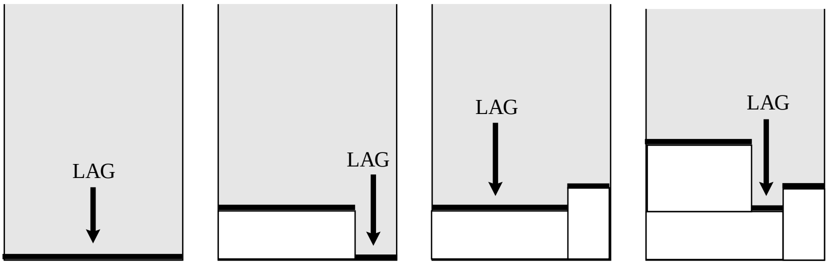


FIGURE 36.9 A heuristic algorithm for the strip packing by Burke et al. [19].

TABLE 36.1 Solution Quality of Heuristic Algorithms for the Strip Packing Problem

	C1	C2	C3	C4	C5	C6	C7
n	17	25	28	49	72	97	196
BL	25	39	33	33	31	34	41
BL-DH	17	68	27	21	18	19	31
BL-DW	18	31	24	18	22	21	29
BLF	14	20	17	15	11	12	10
BLF-DH	11	42	12	6	5	5	4
BLF-DW	11	12	12	5	5	5	5
BKW	12	7	10	4	3	2	2

Computational results are shown in Table 36.1. Each row corresponds to a heuristic algorithm and each column corresponds to a category of instances, where n is the number of items for each instance. In this table, the relative distances in percentage between the optimal and resulting solutions are reported. The computation time for each instance is within 1 second on a PC with an 850 MHz CPU (for BKW) or a 200 MHz CPU (for others). From Table 36.1, we can observe that BLF outperformed BL by up to 25% and that preordering the items by decreasing width or decreasing height for BL and BLF algorithms increased the packing quality by up to 10% compared to random permutations. Moreover, the algorithm by Burke et al. outperformed other algorithms, especially for large instances.

36.5 Metaheuristics for Rectangle Packing

In the last decade, many local search and metaheuristic algorithms for rectangle packing problems have been proposed. Dowsland [24] was one of the early researchers who implemented metaheuristics for rectangle packing problems. Her simulated annealing (SA) algorithm explores both feasible and infeasible (i.e., some items overlap) solutions. During the search, the objective is to reduce the overlapping area. Computational results for small problem instances are reported in Ref. [24].

Let us explain some metaheuristic algorithms based on the permutation coding scheme. These algorithms consist of two phases: (1) find a good permutation using metaheuristics and (2) the decoding algorithm places the items one by one following the permutation order. In Ref. [22], Jakobs proposed a metaheuristic algorithm for the strip packing problem. In this algorithm, he uses a genetic algorithm (GA) to find a good permutation, and places items by using the BL strategy explained in the previous section. He treats not only rectangle packing problems but also irregular packing problems, and reports several computational results. Liu and Teng [23] also proposed a GA algorithm using their BL type decoding algorithm.

In Ref. [20], Hopper and Turton compare the performance of various metaheuristics (multistart local search [MLS], SA, GA, and so on) with two decoding rules on small and large test instances. The computational results reported in Ref. [20] are shown in Table 36.2. The first decoding rule is the BL heuristic proposed by Jakobs [22], and the second one is the BLF strategy proposed by Baker et al. [18]. The stopping criterion for each algorithm is a fixed number of iterations, and the computation time for each instance is about 50,000 times as large as the simple heuristic algorithms (BL and BLF). The representation of this table is similar to that of Table 36.1. From this table, we observe that the performance of the hybrid

TABLE 36.2 Solution Quality of Metaheuristic Algorithms for the Strip Packing Problem

	C1	C2	C3	C4	C5	C6	C7
GA+BL	6	10	8	9	11	15	21
SA+BL	4	7	7	6	6	7	13
MLS+BL	9	18	11	14	14	20	25
GA+BLF	4	7	5	3	4	4	5
SA+BLF	4	6	5	3	3	3	4
MLS+BLF	7	10	7	7	6	7	7

algorithms is strongly dependent on the decoding rule and the instance size. Moreover, it is also reported that certain computation time is needed to attain better solutions with metaheuristics than the solutions obtained by well-designed heuristics such as BLF-DW and BLF-DH.

We now discuss metaheuristic algorithms based on other coding schemes. For the sequence pair representation, Murata et al. [10] proposed an SA algorithm and Imahori et al. [13] proposed an iterated local search (ILS) algorithm. They used metaheuristics to find a good coded solution where each coded solution is evaluated with its own decoding algorithms. Chang et al. [16] and Nakatake et al. [14] proposed SA algorithms using B*-trees and BSG, respectively. One of the advantages of the above algorithms is generality: Imahori et al. [13] incorporated "spatial cost functions" into their algorithms, which was used to handle various types of rectangle packing problems and scheduling problems. Chang et al. [16] and Nakatake et al. [14] designed algorithms that can treat the rectangle packing problem with additional constraints such as preplaced items and soft modules.

Recently, more effective algorithms for rectangle packing problems have been proposed. Lesh et al. [25] proposed a stochastic search variation of the bottom left heuristics for the strip packing problem. Their algorithm outperforms other heuristic and metaheuristic algorithms based on the BL strategy reported in the literature. Furthermore, they incorporated their algorithm in an interactive system that combines the advantages of computer speed and human reasoning. Using the interactive system, they succeeded in producing significantly better solutions than their original algorithm quickly.

Imahori et al. [26] proposed an improved metaheuristic algorithm based on sequence pair representation. Metaheuristic algorithms generate numerous number of coded solutions and evaluate all of them. Hence, the efficiency of metaheuristic algorithms strongly depends on the time complexity of decoding algorithms. Imahori et al. proposed new decoding algorithms to evaluate all coded solutions in various neighborhoods efficiently. As a result, they attained an amortized constant time to evaluate one coded solution in basic neighborhoods.

Bortfeldt [27] proposed a GA for the strip packing problem that works without any encoding of solutions. Instead of using a coding scheme, fully defined layouts are directly manipulated by specific genetic operators. He conducted thorough computational experiments using existing benchmark instances with up to 5000 rectangles, and compared his algorithm with 11 competing methods that were proposed in 1993–2004. He reported that his GA performed best among them.

For more information of the metaheuristic algorithms applied to rectangle packing problems, we refer the reader to articles by Hopper and Turton [20] and Bortfeldt [27].

36.6 Irregular Packing Problem

In this section, we consider the two-dimensional *irregular packing problem,* which has been actively studied in the last decade. The irregular packing problem has many practical applications, for example, the garment, shoe, and shipbuilding industries, and many variants of this problem have been considered in the literature. Among the numerous variants of this problem, the irregular strip packing problem has been studied extensively.

Irregular strip packing problem. We are given n items of arbitrary shapes, and one object (called a strip) with constant width W but variable height H. The objective is to minimize the height H of the strip such that all the items can be packed into the strip.

In this section, we mainly focus on fixed orientation packing. The problem in which items can be rotated (freely or by some fixed degrees) has also been studied in the literature. One of the main differences between rectangle and irregular packing problems is that the intersection test between irregular items is considerably more complex than the case with rectangular items. To overcome this difficulty, some approximation techniques and geometric algorithms have been incorporated into the packing algorithms.

One popular idea for speeding up the intersection test is to represent the items (irregular shapes) approximately. Oliveira and Ferreira [28] proposed two approaches to the irregular strip packing problem, and one of them uses this type of approach. Their approach is based on a raster representation (in other words, bitmap representation) of the irregular shapes to be placed. This approximation allows a quick test overlapping, but suffers from inaccuracy, caused by the approximation inherent in the raster representation. Their another approach uses a polygon-based representation that does not use any approximation technique. Both methods allow overlap in the solutions, and the extent of overlap is penalized by an evaluation function. They try to find a good solution via SA, where the algorithms aim to reduce the overlap to zero.

Okano [29] proposed a heuristic algorithm for the irregular two-dimensional bin packing problem using a scanline representation. He approximates a two-dimensional item with a set of parallel line segments. He also uses a clustering technique; some items are gathered and packed tightly, and then these items are treated as one new item. Okano designed his algorithm for the irregular two-dimensional bin packing problem; however, his technique is also useful for treating the irregular strip packing problem. He conducted computational experiments with real instances from a shipbuilding company, and reported that the quality of the resulting layouts was sufficiently high for practical use.

Jakobs [22] used another type of approximation scheme; for all irregular shapes, the minimum bounding rectangles of the given items are calculated and his algorithm treats these rectangles instead of the original shapes. For these shapes, a good placement is computed by his BL decoding algorithm and GA techniques. After finding a good placement for the rectangles, the algorithm replaces rectangles with the original irregular shapes, and computes a better placement of the irregular items. He reported computational results for this algorithm. Jakobs also discussed an idea of clustering several shapes, and finding the minimum bounding rectangle of several items.

Dighe and Jakiela [30] proposed an algorithm for the irregular strip packing problem, which is based on a clustering method with a tree structure. Their algorithm uses a tree structure to represent a solution: The leaves of a tree correspond to items, and clustering operations are applied to items from the leaves to the root of the tree. To find a good coded solution (i.e., tree), they utilized GA. Dighe and Jakiela created new test instances and conducted computational experiments on these instances.

One of the most popular geometric techniques used for the intersection test is *no-fit polygon*. The concept of no-fit polygon was introduced by Art [31] in 1966, who used the term "shape envelope" to describe the positions where two items can be placed without intersection. Albano and Sapuppo [32] proposed an algorithm to solve the irregular strip packing problem with this geometric technique. This was the first paper that used the term "no-fit polygon." This concept is also known as Minkowski sums, and is utilized in various fields such as motion planning for polygonal robots.

The no-fit polygon of item j relative to item i ($NFP_{i,j}$) is the set of all loci of the reference point of item j (denoted R_j) such that items i and j have a common point when the position of item i is fixed (each item is considered as the set of points on the boundary and inside it). If both items i and j are convex, the boundary of $NFP_{i,j}$ is the trace of R_j when item j slides along the boundary of item i. See Figure 36.10 for an example of the no-fit polygon $NFP_{i,j}$ of convex polygons i and j. From the definition of no-fit polygons, it follows that

- if the reference point R_j of item j is placed in the interior of $NFP_{i,j}$, then item j overlaps item i;
- if R_j is placed on the boundary of $NFP_{i,j}$, then item j touches item i;
- if R_j is placed in the exterior of $NFP_{i,j}$, then item j neither overlaps nor touches item i.

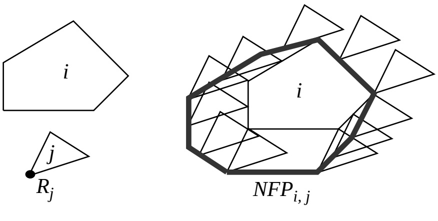


FIGURE 36.10 An example of the no-fit polygon $NFP_{i,j}$ of convex items i and j.

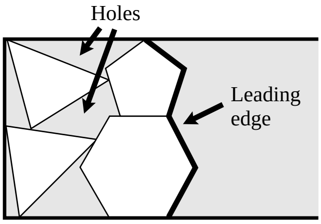


FIGURE 36.11 The leading edge and holes in a layout of irregular shapes.

The problem of finding the relative position of two polygons is transformed into a simpler problem of finding the relative position of one point and one polygon. To achieve a nonoverlapping compact layout, each item should have its reference point on the boundary of at least one no-fit polygon and in the exterior of all the other no-fit polygons.

Albano and Sapuppo [32] proposed an algorithm to solve the irregular strip packing problem using no-fit polygons. They approached the problem using a BL algorithm, which utilized the no-fit polygon to reduce the geometric complexity of the packing process. Their algorithm places each item one by one at the right frontier called the leading edge of the current layout only, that is, without hole filling capabilities. See Figure 36.11 for an example of the leading edge and holes in a layout. Blazewicz et al. [33] presented an extension of the work performed by Albano and Sapuppo [32]. Their method is an extension of the BLF algorithm; that is, their approach attempts to fill holes in the existing layout before attempting to place an item on the leading edge. Their algorithm utilizes the tabu search technique to produce moves from one solution to another.

Oliveira et al. [34] also tackled the irregular strip packing problem using no-fit polygons. Their algorithm places all small items one by one at a nonoverlapping position touching at least one item already placed. Several criteria to choose the next item to place and its orientation were proposed (in this article, they treated a problem such that each item can be rotated by some fixed degrees). Different evaluation functions were also proposed to evaluate partial solutions and to decide the position of each item. A total of 126 variants of the algorithm, generated by the complete set of combinations of criteria and evaluation functions, were computationally compared. In their computational experiments, they solved several types of test instances; test instances from a fabric cutting company and a test instance generated by Blazewicz et al. [33]. Oliveira et al. compared their results against an implementation of Albano and Sapuppo's algorithm [32], and against results from Blazewicz et al. [33]. In some cases, their new algorithm generated better solutions than the best known solutions in the literature; more precisely, their algorithm generated solutions that ranged from 6.2% better to 4% worse than the best known results.

Gomes and Oliveira [35] developed shape-ordering heuristics for an extended irregular packing algorithm similar to that given by Oliveira et al. [34]. The algorithm is improved by the introduction of the inner-fit rectangle, which is derived from the concept of no-fit polygon and represents the feasible set of points for placing a new polygon inside the object. In addition to this extension of geometric techniques, the paper introduces a 2-exchange heuristic for manipulating a permutation that specifies the order of placing items one by one. They generated some initial permutations with various criteria, for example, random, decreasing order of area, decreasing order of the longest length of items, and improved them using the 2-exchange heuristic over a number of iterations. In [35], they conducted thorough computational experiments and compared the proposed algorithm with existing algorithms. They improved almost all best known solutions for well-known benchmark instances (except for an instance called SHAPE0).

Gomes and Oliveira [36] also developed a hybrid algorithm of simulated annealing with linear programming (LP) technique to solve the irregular strip packing problem. In this algorithm, a neighborhood structure based on the exchange of items on the layout was used for SA. For a given layout (which is neither feasible nor tight), they solved LP to locally optimize the layout. Computational tests were conducted using 15 benchmark instances that are commonly used in the literature, and the best results published so far were improved for all instances by their new algorithm.

Burke et al. [37] proposed a heuristic algorithm for the irregular strip packing problem with new shape overlap resolution techniques applied to the given shapes directly (i.e., without reference to no-fit polygons). In this article, they treated not only polygons (i.e., shapes with line representation) but also shapes that incorporate circular arcs and holes, and proposed overlap resolution techniques for line & line, line & arc, and arc & arc. Items are placed one by one according to a permutation (coded solution) with the BLF strategy, and tabu search is used to find a good permutation. They conducted computational experiments on 26 existing benchmark instances and 10 new test instances with items having circular arcs and holes. Their technique produced 25 new best solutions for the 26 existing benchmark instances; most of them were found within 5 minutes on a PC with a 2 GHz CPU.

36.7 Conclusions

In this chapter, we surveyed practical algorithms for the two-dimensional rectangle packing problem and the irregular packing problem, both of which have many industrial applications. For the rectangle packing problem, we first introduced some coding schemes (in other words, how to represent a solution) such as permutation, binary tree, sequence pair, and BSG. We then explained various heuristic and metaheuristic algorithms, most of which are based on these coding schemes. For some representative algorithms, we reported computational results on benchmark instances and compared them. The irregular packing problem has also been studied extensively in the last decade. The main difference between rectangle and irregular packing problems is that the intersection test between irregular items is considerably more complex. To overcome this difficulty, some different types of methodologies have been proposed; no-fit polygon is one of the representative and effective ones. We explained some heuristic and metaheuristic algorithms based on these techniques for the irregular packing problem.

The survey in this chapter is by no means comprehensive, but we hope this gives valuable information to the readers who are interested in devising practical algorithms for cutting and packing problems. Fortunately, there have been many survey papers (30 or more in these 20 years) on cutting and packing problems; for example, Dyckhoff [1] and Wäscher et al. [2] presented typologies of cutting and packing problems and categorized existing literature, and Hopper and Turton [20] investigated heuristic and metaheuristic algorithms for the rectangle packing problem. Chapters 32–35, 78, and 79 of this handbook may also be useful for readers who are interested in related topics.

References

[1] Dyckhoff, H., A typology of cutting and packing problems, *Eur. J. Oper. Res.,* 44, 145, 1990.

[2] Wäscher, G., Haußner, H., and Schumann, H., An Improved Typology of Cutting and Packing Problems, Working paper 24, Faculty of Economics and Management, Guericke University Magdeburg, 2004.

[3] Lodi, A., Martello, S., and Vigo, D., Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, *INFORMS J. Comput.,* 11, 345, 1999.

[4] Wu, Y. L., Huang, W., Lau, S., Wong, C. K., and Young, G. H., An effective quasi-human based heuristic for solving the rectangle packing problem, *Eur. J. Oper. Res.,* 141, 341, 2002.

[5] Gilmore, P. C. and Gomory, R. E., Multistage cutting stock problems of two and more dimensions, *Oper. Res.,* 13, 94, 1965.

[6] Valdés, R. A., Parajón, A., and Tamarit, J. M., A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems, *Comput. Oper. Res.,* 29, 925, 2002.

[7] Vanderbeck, F., A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem, *Manage. Sci.,* 47, 864, 2001.

[8] Morabito, R. and Morales, S., A simple and effective recursive procedure for the manufacturer's pallet loading problem, *J. Oper. Res. Soc.,* 49, 819, 1998.

[9] Preas, B. T. and van Cleemput, W. M., Placement algorithms for arbitrarily shaped blocks, *Proc. DAC,* 1979, p. 474.

[10] Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y., VLSI module placement based on rectangle-packing by the sequence-pair, *IEEE Trans. CAD,* 15, 1518, 1996.

[11] Takahashi, T., An Algorithm for Finding a Maximum-Weight Decreasing Sequence in a Permutation, Motivated by Rectangle Packing Problem, Technical report of the IEICE, VLD96, 1996, 31.

[12] Tang, X., Tian, R., and Wong, D. F., Fast evaluation of sequence pair in block placement by longest common subsequence computation, *IEEE Trans. CAD,* 20, 1406, 2001.

[13] Imahori, S., Yagiura, M., and Ibaraki, T., Local search algorithms for the rectangle packing problem with general spatial costs, *Math. Prog.,* 97, 543, 2003.

[14] Nakatake, S., Fujiyoshi, K., Murata, H., and Kajitani, Y., Module packing based on the BSG-structure and IC layout applications, *IEEE Trans. CAD,* 17, 519, 1998.

[15] Guo, P. N., Takahashi, T., Cheng, C. K., and Yoshimura, T., Floorplanning using a tree representation, *IEEE Trans. CAD,* 20, 281, 2001.

[16] Chang, Y. C., Chang, Y. W., Wu, G. M., and Wu. S. W., B*-trees: A new representation for non-slicing floorplans, *Proc. DAC,* 2000, p. 458.

[17] Sakanushi, K., Kajitani, Y., and Mehta, D. P., The quarter-state-sequence floorplan representation, *IEEE Trans. Circuits Sys.,* 50, 376, 2003.

[18] Baker, B. S., Coffman Jr., E. G., and Rivest, R. L., Orthogonal packing in two dimensions, *SIAM J. Comput.,* 9, 846, 1980.

[19] Burke, E. K., Kendall, G., and Whitwell, G., A new placement heuristic for the orthogonal stock-cutting problem, *Oper. Res.,* 52, 655, 2004.

[20] Hopper, E. and Turton, B. C. H., An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem, *Eur. J. Oper. Res.,* 128, 34, 2001.

[21] Chazelle, B., The bottom-left bin-packing heuristic: An efficient implementation, *IEEE Trans. Comput.,* 32, 697, 1983.

[22] Jakobs, S., On genetic algorithms for the packing of polygons, *Eur. J. Oper. Res.,* 88, 165, 1996.

[23] Liu, D. and Teng, H., An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles, *Eur. J. Oper. Res.,* 112, 413, 1999.

[24] Dowsland, K., Some experiments with simulated annealing techniques for packing problems, *Eur. J. Oper. Res.,* 68, 389, 1993.

[25] Lesh, N., Marks, J., McMahon, A., and Mitzenmacher, M., New heuristic and interactive approaches to 2D rectangular strip packing, *ACM J. Exp. Algorithmics,* 10(1–2), 1, 2005.

[26] Imahori, S., Yagiura, M., and Ibaraki, T., Improved local search algorithms for the rectangle packing problem with general spatial costs, *Eur. J. Oper. Res.,* 167, 48, 2005.

[27] Bortfeldt, A., A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces, *Eur. J. Oper. Res.*, 172, 814, 2006.

[28] Oliveira, J. F. and Ferreira, J. S., Algorithms for nesting problems, in *Applied Simulated Annealing,* Vidal, R. V., Ed., Springer-Verlag, Berlin, 1993, p. 255.

[29] Okano, H., A scanline-based algorithm for the 2D free-form bin packing problem, *J. Oper. Res. Soc. Jpn.*, 45, 145, 2002.

[30] Dighe, R. and Jakiela, M. J., Solving pattern nesting problems with genetic algorithms employing task decomposition and contact detection, *Evol. Comput.*, 3, 239, 1996.

[31] Art, R. C., An Approach to the Two Dimensional Irregular Cutting Stock Problem, Technical report, 36-Y08, IBM Cambridge Scientific Center Report, 1966.

[32] Albano, A. and Sapuppo, G., Optimal allocation of two-dimensional irregular shapes using heuristic search methods, *IEEE Trans. Syst., Man Cybern.*, 10, 242, 1980.

[33] Blazewicz, J., Hawryluk, P., and Walkowiak, R., Using a tabu search for solving the two-dimensional irregular cutting problem, *Ann. Oper. Res.,* 41, 313, 1993.

[34] Oliveira, J. F., Gomes, A. M., and Ferreira, J. S., TOPOS —A new constructive algorithm for nesting problems, *OR Spektrum,* 22, 263, 2000.

[35] Gomes, A. M. and Oliveira, J. F., A 2-exchange heuristic for nesting problems, *Eur. J. Oper. Res.,* 141, 359, 2002.

[36] Gomes, A. M. and Oliveira, J. F., Solving irregular strip packing problems by hybridising simulated annealing and linear programming, *Eur. J. Oper. Res.*, 171, 811, 2006.

[37] Burke, E. K., Hellier, R., Kendall, G., and Whitwell, G., A new bottom-left-fill heuristic algorithm for the 2D irregular packing problem, *Oper. Res.*, 54, 587, 2006.

37

A Generic Primal-Dual Approximation Algorithm for an Interval Packing and Stabbing Problem

Sofia Kovaleva
University of Maastricht

Frits C. R. Spieksma
Catholic University of Leuven

37.1 Introduction

Packing and stabbing (or covering) problems are two basic problems in combinatorial optimization. Apart from the fact that they arise in many practical applications, investigating these problems improves our understanding of fundamental issues in combinatorial optimization. Admittedly, the generality of a packing or a stabbing problem has its price: When it comes to solving such a problem, and when one insists that an optimal solution is produced for all instances, one needs to accept that, for some instances, large running times are unavoidable. Moreover, when one restricts oneself to polynomial-time algorithms, only very weak statements concerning the quality of the solutions found can be made (see, e.g., Chapter 1, Ausiello et al. [1] or Vazirani [2] for an introduction and terminology).

Here, we focus on a special case of these two problems. On the one hand, by studying a (geometric) special case, we enter the world of polynomial-time algorithms that admit a constant performance guarantee; on the other, this special case is still general enough to admit a variety of applications.

As an appetizer, consider the following two questions dealing with intervals on the line. Imagine that n intervals $(l_i, r_i]$, $i = 1, \ldots, n$, etc. on the line are given; further, we say that two intervals are *disjoint* when their intersection is empty, interval i *contains* point p when $l_i < p \leq r_i$, and interval i is *stabbed* by point p when interval i contains p.

Question 1. Select as many pairwise disjoint intervals as possible.

Question 2. Stab every interval at least once using as few points as possible.

These two questions are, in fact, easy to answer. Indeed, it is well known that the following rule does the job. After sorting the intervals in nondecreasing order with respect to the r_i's, repeat the following instructions iteratively: (i) select the remaining interval with the smallest r_i, (ii) discard all intervals containing r_i, and (iii) add point r_i to the set of points. Essentially, this rule solves an independent set problem, and a clique cover problem, in an interval graph.

This chapter deals with generalizations of this setting. We consider the case when the set of intervals is partitioned into, say m, subsets such that at most one interval of a subset can be selected (while still requiring disjointness). Such a subset of intervals is sometimes referred to as a *job*. Even more generally, we consider the problems that arise when a given number of intervals from a subset can be selected, and when the disjointness restriction is replaced by allowing a given number of intervals to intersect each other. Also, weights for the intervals are considered.

In the sequel we formulate precisely the problems we consider, describe previous research, and explain the setup of this chapter. Our main focus is on the description of a generic primal-dual algorithm for a weighted set packing problem (WSP), and how this method works for our particular geometric setting. This chapter is based on a part of the Ph.D. thesis of Kovaleva [3].

Preliminaries

We use an underlying grid to formulate our problems. Given is a grid consisting of t columns, numbered consecutively from left to right, and m numbered rows. Furthermore, given is a set of intervals $I = \{1, 2, \ldots, n\}$ lying on the rows of the grid. An interval $i \in I$ is specified by the triple l_i, r_i, ρ_i, where l_i, r_i are the indices of the left- and the rightmost columns intersecting interval i and ρ_i is the index of the row where interval i lies. For each column c, row r, and interval i we are given positive integral parameters v_c, u_r, and p_i, respectively, referred to as the column, row, and interval capacities. For each interval i we are given a positive integral parameter w_i referred to as the interval demand. We assume that the intervals are ordered according to nondecreasing r_i. We refer to the family of inputs that arise in this way as Job Interval instances (JI). Thus, an instance $\mathcal{I}$ of JI can be seen as a collection of numbers t, m, for each $i \in I : l_i, r_i, \rho_i, p_i, w_i$, for each $c = 1, \ldots, t : v_c$, and for each $r = 1, \ldots, m : u_r$.

We use the following terminology in the sequel: Column c is said to *stab* interval i if column c intersects interval i, that is, $l_i \leq c \leq r_i$. Also, row ρ_i is said to *stab* interval i if interval i lies on row ρ_i.

For each instance $\mathcal{I}$ of JI we formulate the following two problems:

Job Interval Packing Problem (JIP)

For each interval $i \in I$ specify an integral multiplicity x_i such that

- *it does not exceed the interval capacity p_i,*
- *for each column c the sum of multiplicities of the intervals stabbed by column c does not exceed the column capacity v_c,*
- *for each row r the sum of multiplicities of the intervals stabbed by row r does not exceed the row capacity u_r,*
- *the total demand $\sum w_i x_i$ is maximized.*

Job Interval Stabbing Problem (JIS)

For each column c, row r, and interval i specify integral multiplicities y_c, z_r, and s_i respectively, such that

- *for each interval i the sum of the multiplicities of the columns stabbing interval i plus the multiplicity of the row stabbing interval i plus the multiplicity of interval i equals at least its demand w_i,*
- *the total capacity $\sum_{c=1}^{t} v_c y_c + \sum_{r=1}^{m} u_r z_r + \sum_{i=1}^{n} p_i s_i$ is minimized.*

One may interpret the interval demands w_i as interval weights in JIP and the column, row and interval capacities v_c, u_r, p_i as column, row, and interval weights in JIS. However, in this chapter we refer to these parameters as demands and capacities to keep the terminology uniform for both problems.

In this chapter we describe a generic primal-dual algorithm and show how it can be applied to two special cases of JIP and JIS. The first special case deals with instances where all the column, row, and interval capacities are unit, that is, $v_c = u_r = p_i = 1$, $\forall c, r, i$. We refer to the family of inputs satisfying this condition as *JI with unit capacities.* The second special case assumes that all the interval demands are unit, that is, $w_i = 1$, $\forall i = 1, \ldots, n$. The family of these inputs is referred to as *JI with unit demands.*

Below we give natural Integer Linear Programming (ILP) formulations of JIP and JIS:

$$\text{JIP: Maximize} \quad \sum_{i=1}^{n} w_i x_i \tag{37.1}$$

$$\text{subject to} \quad \sum_{i:\rho_i=r} x_i \le u_r \quad \forall r = 1, \ldots, m \tag{37.2}$$

$$\sum_{i:c\in[l_i,r_i]} x_i \le v_c \quad \forall c = 1, \ldots, t \tag{37.3}$$

$$x_i \le p_i \quad \forall i = 1, \ldots, n \tag{37.4}$$

$$x_i \in \mathbb{Z}_+^1 \quad \forall i = 1, \ldots, n \tag{37.5}$$

$$\text{JIS: Minimize} \quad \sum_{c=1}^{t} v_c y_c + \sum_{r=1}^{m} u_r z_r + \sum_{i=1}^{n} p_i s_i \tag{37.6}$$

$$\text{subject to} \quad z_{\rho_i} + \sum_{c\in[l_i,r_i]} y_c + s_i \ge w_i \quad \forall i = 1, \ldots, n \tag{37.7}$$

$$z_r, y_c, s_i \in \mathbb{Z}_+^1 \quad \forall r, c, i \tag{37.8}$$

Observe that for any instance $\mathcal{I}$ of JI the Linear Programming (LP) relaxations of these ILP formulations (which arise when we substitute the integrality constraints [37.5] and [37.8] by nonnegativity constraints $x_i \ge 0$, $\forall i = 1, \ldots, n$ and $z_r, y_c, s_i \ge 0, \forall r, c, i$, respectively) constitute a primal-dual pair of LP problems. It follows then from the strong duality theorem for linear programming that for any instance $\mathcal{I}$, the LP relaxations of JIP and JIS have the same optimum value, which we denote by $LP(\mathcal{I})$, and thus the following holds:

$$JIP(\mathcal{I}) \le LP(\mathcal{I}) \le JIS(\mathcal{I}) \tag{37.9}$$

where $JIP(\mathcal{I})$ and $JIS(\mathcal{I})$ are the optimum values of JIP and JIS for $\mathcal{I}$, respectively.

We say that problems JIP and JIS are weakly dually related and establish the following result:

Lemma 37.1 (Weak duality lemma for JIP and JIS)

For any instance $\mathcal{I}$ of JI, for any feasible solution to JIP with value $JIP^{feas}(\mathcal{I})$ and any feasible solution to JIS with value $JIS^{feas}(\mathcal{I})$ holds:

$$JIP^{feas}(\mathcal{I}) \le JIS^{feas}(\mathcal{I}).$$

Proof

Follows from (37.9). □

Previous Research

JIP and its special cases have received a great deal of attention in the literature. Its applications, mentioned by different authors, include printed circuit board assembly, combinatorial auctions, satellite photography, computational biology, throughput scheduling [4–7]. JIP is MAX SNP-hard even if all the parameters are unit ([7]; this implies that there is no polynomial-time approximation scheme for JIP unless $\mathcal{P} = \mathcal{NP}$). A greedy 1/2-approximation algorithm for JIP with unit capacities and demands (i.e., $u_r = v_c = p_i = w_i = 1$, $\forall c, r, i$) is described in Spieksma [7]. A better approximation guarantee for this case has been found by Chuzhoy et al. [8]. They present an algorithm with performance guarantee $(e-1)/e - \varepsilon \approx 0.63 - \varepsilon$, for any $\varepsilon > 0$, exploiting a sophisticated technique involving randomized LP-rounding. Bar-Noy et al. [6] describe a $(1/2 - \varepsilon)$-approximation algorithm for JIP with unit capacities (i.e., $u_r = v_c = p_i = 1$, $\forall c, r, i$) using an LP-rounding technique. One can easily generalize this algorithm to JIP (with arbitrary capacities and demands) and, using the ideas introduced by Calinescu et al. [9], improve the approximation factor to 1/2. The resulting 1/2-approximation algorithm (which involves solving an LP problem) has to our knowledge so far the highest approximation factor for JIP. Combinatorial 1/2-approximation algorithms for JIP with unit capacities are described by Berman and DasGupta [4] and Bar-Noy et al. [10]. In Ref. [5] the performance of greedy algorithms for JIP with constant column capacity and unit row capacity (i.e., $v_c = v,\ \forall c,\ u_r = 1,\ \forall r$) is investigated using competitive analysis.

So far JIS has not been as extensively studied as JIP. Applications of JIS that have been mentioned in the literature include military and medical applications [11]. JIS is also MAX SNP-hard even if all the parameters are unit [12]. Recently, Kovaleva and Spieksma [13] describe a $\frac{e}{e-1}$-approximation algorithm for JIS with unit demand. This algorithm is based on solving the linear programming relaxation of constraints (37.6)–(37.8), and rounding this solution to an integral one. Earlier work is described in Gaur et al. [14] who consider the so-called *rectangle stabbing* problem, which generalizes JIS with unit demands to the case where rectangles intersecting several rows of the grid are given instead of intervals; their work implies a 2-approximation for JIS with unit demand. Hassin and Megiddo [11] describe a combinatorial $2 - \frac{1}{K+1}$-approximation algorithm for the case of JIS with unit capacities and demands, where each interval is intersected by exactly K columns, and a 2-approximation for a slightly more general case, when each interval is intersected by the same number of columns.

Outline of the Chapter

Section 37.2 is devoted to JI with unit capacities. We first describe a generic primal-dual algorithm, called Local Covering, for the WSP (Section 37.2.1) and then specify a setup of Local Covering for JIP with unit capacities, yielding algorithm ALG1 (Section 37.2.2). We also give conditions which guarantee that a particular setup of Local Covering yields a constant factor approximation algorithm. This algorithm is in fact a primal-dual interpretation of the *Opportunity Cost* algorithm described by Akcoglu et al. in [15], which is in turn based on the local-ratio technique, introduced by Bar-Yehuda and Even [16] and later elaborated upon by Bar-Noy et al. [10].

A distinctive feature of the primal-dual algorithm ALG1 described in Section 37.2.2 is that it simultaneously delivers two feasible solutions, one for JIP and one for JIS with unit capacities. Moreover, we show that their values are within a factor of 2 of each other. The weak duality relation between JIP and JIS implies then that these solutions are respectively a 1/2-approximation for JIP with unit capacities and a 2-approximation for JIS with unit capacities. This analysis is tight, that is, the approximation guarantees cannot be improved by carrying out a better analysis of the algorithm.

Note that, when viewed as a 1/2-approximation algorithm for JIP, ALG1 behaves similar to the algorithms described previously by Berman and DasGupta [4] and Bar-Noy et al. [10].

In Section 37.3 we consider JI with unit demands. It is easy to see that JIS with unit demands is a special case of the the weighted hitting set problem. Following the framework of the generic *primal-dual algorithm with reverse delete step* described in Ref. [17] for the weighted hitting set problem, we develop a primal-dual approximation algorithm ALG2. Similar to ALG1 it returns two feasible solutions, one for JIS and the other for JIP with unit demands. Again, we show that their values are within a factor of 2 of each other. This makes ALG2 a 1/2-approximation algorithm for JIP and a 2-approximation algorithm to JIS with unit demands.

Note that the algorithms described here are combinatorial and do not require solving a linear program.

37.2 The Case of Unit Capacities

37.2.1 The Local Covering Algorithm for the Weighted Set Packing Problem

Consider the WSP: *given is a collection $\mathcal{S}$ of subsets of a finite ground set $\mathcal{E}$. For each subset $s \in \mathcal{S}$ a nonnegative weight w_s is given. Find a maximum-weight collection A of disjoint subsets from $\mathcal{S}$.*

Below we give a natural ILP formulation of WSP, where we use the characteristic vector of A, $\chi \in \{0, 1\}^{|\mathcal{S}|}$, as a vector of decision variables:

$$\text{WSP:} \quad \text{Maximize} \quad \sum_{s \in \mathcal{S}} w_s \chi_s \tag{37.10}$$

$$\text{subject to} \quad \sum_{s : e \in s} \chi_s \leq 1 \quad \forall e \in \mathcal{E} \tag{37.11}$$

$$\chi_s \in \{0, 1\} \quad \forall s \in \mathcal{S} \tag{37.12}$$

By substituting nonnegativity constraints $\chi_s \geq 0 \ \ \forall s \in \mathcal{S}$ for integrality constraints (37.12), we obtain the LP relaxation of this formulation.

Let us also introduce the dual linear problem to this LP relaxation (here we use $\gamma \in \mathbb{R}_+^{|\mathcal{E}|}$ for a vector of dual variables):

$$\text{Minimize} \quad \sum_{e \in \mathcal{E}} \gamma_e \tag{37.13}$$

$$\text{subject to} \sum_{e \in s} \gamma_e \geq w_s \quad \forall s \in \mathcal{S} \tag{37.14}$$

$$\gamma_e \geq 0 \qquad \forall e \in \mathcal{E} \tag{37.15}$$

For ease of discussion let us introduce some terminology. We say that

- an element $e \in \mathcal{E}$ and a subset $s \in \mathcal{S}$ are *incident* to each other if $e \in s$;
- the *coverage* of a subset $s \in \mathcal{S}$ by a vector $\gamma \in \mathbb{R}_+^{|\mathcal{E}|}$ is a value equal to the sum of the elements of γ corresponding to the elements e belonging to s, that is, $\sum_{e \in s} \gamma_e$;
- a subset $s \in \mathcal{S}$ is *covered* if its coverage equals at least w_s. Otherwise, we say that s is *violated*;
- a subset s_1 is a *neighbor* of a subset s_2 if they share a common element;
- $N(s)$ is the set of all the neighbors of s. Note that $s \in N(s)$;
- a feasible solution χ to the ILP formulation (37.10)–(37.12) as well as the corresponding set $A \in \mathcal{S}$ is a *feasible packing*;
- a feasible solution γ to the LP formulation (37.13)–(37.15) is a *feasible covering*.

Definition 37.1

For a given vector $\gamma \in \mathbb{R}_+^{|\mathcal{E}|}$ and a subset s^* let us call a vector $\delta \in \mathbb{R}_+^{|\mathcal{E}|}$ satisfying: $\sum_{e \in s} \delta_e \geq \min(w_s - \sum_{e \in s} \gamma_e, w_{s^*} - \sum_{e \in s^*} \gamma_e)$, $\forall s \in N(s^*)$, a local covering for γ in s^*.

Let us now describe a generic primal-dual algorithm local covering for WSP. The framework of the algorithm is the following: Initially, all the dual variables γ_e are zero and A is empty. Until γ becomes a feasible covering do

- select (some) subset s, violated by the current γ, and push it on a stack,
- construct a local covering δ for γ in s,
- increment vector γ by the the values of vector δ.

When γ becomes a feasible covering, pop the subsets from the stack iteratively and each time add a subset to A if this does not violate the feasibility of A.

Figure 37.1 gives a formal description of local covering (notice that $\delta \in \mathbb{R}_+^{|\mathcal{E}|}$, $\Delta \in \mathbb{R}_+$, $l \in \mathbb{N}$).

Observe that it is not exactly specified in the algorithm how s^l and δ are selected. The description of these selection procedures is left to a particular setup. It does not make sense to discuss implementation and efficiency of the algorithm in such a general setting. Let us only establish the following result:

```
1. $\gamma \leftarrow \mathbf{0}$

2. $A \leftarrow \emptyset$

3. $l \leftarrow 0$

4. Until $\gamma$ is a feasible covering
     $l \leftarrow l + 1$
     Select $s^l$ such that $\sum_{e \in s^l} \gamma_e < w_{s^l}$
     $\Delta \leftarrow w_{s^l} - \sum_{e \in s^l} \gamma_e$
     Find $\delta \in \mathbb{R}_+^{|\mathcal{E}|}$ s. t.: $\sum_{e \in s} \delta_e \geq \min(w_s - \sum_{e \in s} \gamma_e, \Delta)$, $\forall s \in N(s^l)$
     $\gamma \leftarrow \gamma + \delta$

5. For $l$ downto 1
     if $A \cup \{s^l\}$ is feasible then $A \leftarrow A \cup \{s^l\}$

6. Return $A$ (and $\gamma$)
```

FIGURE 37.1 Algorithm Local Covering.

Theorem 37.1

For any instance $\mathcal{I}$ of WSP, the set A and the vector γ returned by local covering are a feasible packing and a feasible covering respectively. Moreover, if Δ and δ found by the algorithm at each iteration satisfy:

$$\sum_{e \in \mathcal{E}} \delta_e \leq \beta \cdot \Delta$$

then

$$\sum_{e \in \mathcal{E}} \gamma_e \leq \beta \sum_{s \in A} w_s$$

Proof

The feasibility of A and γ is obvious. Let us establish the relation between their values. Let p be the number of iterations made by the algorithm at step 4. Observe that p is at most $|\mathcal{S}|$ since at each iteration the number of violated subsets decreases by at least 1 and the iterations stop when there is no more violated subset. Let Δ^l, δ^l and γ^l $(l = 1, \ldots, p)$ be the values of Δ, vector δ, and vector γ respectively at the end of the lth iteration of step 4. Let γ^0 be a zero vector. So we have $\gamma^l = \gamma^{l-1} + \delta^l$, $\forall l = 1, \ldots, p$. The condition of the theorem can be written as

$$\sum_{e \in \mathcal{E}} \delta_e^l \leq \beta \cdot \Delta^l \quad \forall l = 1, \ldots, p$$

Further, let A^q be the state of set A at the end of the loop of the cycle at step 5, corresponding to $l = q$. Let A^{p+1} be $\emptyset$. Then we have $\emptyset = A^{p+1} \subseteq A^p \subseteq \cdots \subseteq A^1$.

We show that for each $l = 1, \ldots, p + 1$

$$\sum_{e \in \mathcal{E}} \left(\gamma_e^p - \gamma_e^{l-1}\right) \leq \beta \sum_{s \in A^l} \left(w_s - \sum_{e \in s} \gamma_e^{l-1}\right) \tag{37.16}$$

Then for $l = 1$, since γ^0 is a zero vector and A^1 is the set A returned by the algorithm, we obtain the result of the theorem.

We use induction on $l = p + 1, \ldots, 1$. The basis of induction is $l = p + 1$. Then inequality (37.16) trivially holds, since $A^{p+1} = \emptyset$. Suppose the inequality is proven for $l = j + 1$. Let us prove it for $l = j$. So, we have

$$\sum_{e \in \mathcal{E}} \left(\gamma_e^p - \gamma_e^j\right) \leq \beta \sum_{s \in A^{j+1}} \left(w_s - \sum_{e \in s} \gamma_e^j\right) \tag{37.17}$$

and the theorem will be established if we show that

$$\sum_{e\in\mathcal{E}}\left(\gamma_e^p-\gamma_e^{j-1}\right)\le\beta\sum_{s\in A^j}\left(w_s-\sum_{e\in s}\gamma_e^{j-1}\right)\tag{37.18}$$

First, let us show that the following inequality holds:

$$\sum_{s\in A^j}\left(w_s-\sum_{e\in s}\gamma_e^{j-1}\right)\ge\sum_{s\in A^{j+1}}\left(w_s-\sum_{e\in s}\gamma_e^{j}\right)+\Delta^j\tag{37.19}$$

There are two cases possible: either $A^j=A^{j+1}$ or $A^j=A^{j+1}\cup\{s^j\}$.

Consider the first case. Suppose $A^j=A^{j+1}$. Clearly, this case is only possible if A^{j+1} contains a neighbor of subset s^j, say subset s^q, that is, $s^q\in N(s^j)$. Consider γ^j, it is equal to $\gamma^{j-1}+\delta^j$, where δ^j satisfies

$$\sum_{e\in s}\delta_e^j\ge\min\left(w_s-\sum_{e\in s}\gamma_e^{j-1},\Delta^j\right),\quad\text{for all } s\in N(s^j)$$

and in particular for s^q, that is, $\sum_{e\in s^q}\delta_e^j\ge\min\left(w_{s^q}-\sum_{e\in s^q}\gamma_e^{j-1},\ \Delta^j\right)$.

Assume

$$\sum_{e\in s^q}\delta_e^j\ge\min\left(w_{s^q}-\sum_{e\in s^q}\gamma_e^{j-1},\Delta^j\right)=w_{s^q}-\sum_{e\in s^q}\gamma_e^{j-1}$$

Then

$$\sum_{e\in s^q}\gamma_e^j=\sum_{e\in s^q}\gamma_e^{j-1}+\sum_{e\in s^q}\delta_e^j\ge w_{s^q}$$

This means that subset s^q is covered by γ^j and therefore also by $\gamma^{j+1},\ldots,\gamma^p$, since $\gamma^j\le\gamma^{j+1}\le\cdots\le\gamma^p$. Therefore, subset s^q is not violated at iteration j or later and cannot be selected and pushed on the stack during iterations $j+1,\ldots,p$ of step 4, which contradicts with $s^q\in A^{j+1}$. Therefore, the only possible case is that

$$\sum_{e\in s^q}\delta_e^j\ge\min\left(w_{s^q}-\sum_{e\in s^q}\gamma_e^{j-1},\Delta^j\right)=\Delta^j$$

Now, using the assumption $A^j=A^{j+1}$, the facts that $\gamma^{j-1}=\gamma^j-\delta^j$ and that there exists $s^q\in A^{j+1}$ such that the above inequality holds, we can rewrite the left-hand side of inequality (37.19) as

$$\begin{aligned}\sum_{s\in A^{j+1}}\left(w_s-\sum_{e\in s}\gamma_e^{j-1}\right)&=\sum_{s\in A^{j+1}}\left(w_s-\sum_{e\in s}\gamma_e^{j}+\sum_{e\in s}\delta_e^j\right)\\&=\sum_{s\in A^{j+1}}\left(w_s-\sum_{e\in s}\gamma_e^{j}\right)+\sum_{s\in A^{j+1}}\sum_{e\in s}\delta_e^j\ge\sum_{s\in A^{j+1}}\left(w_s-\sum_{e\in s}\gamma_e^{j}\right)+\Delta^j\end{aligned}$$

which proves inequality (37.19) in the case $A^j=A^{j+1}$.

Consider now the case $A^j=A^{j+1}\cup\{s^j\}$.

Using the fact that $\Delta^j=w_{s^j}-\sum_{e\in s^j}\gamma_e^{j-1}$ and $\gamma^{j-1}\le\gamma^j$, rewrite the left-hand side of inequality (37.19) as

$$\sum_{s\in A^{j+1}}\left(w_s-\sum_{e\in s}\gamma_e^{j-1}\right)+\left(w_{s^j}-\sum_{e\in s^j}\gamma_e^{j-1}\right)\ge\sum_{s\in A^{j+1}}\left(w_s-\sum_{e\in s}\gamma_e^{j}\right)+\Delta^j$$

which proves inequality (37.19) in the case $A^j=A^{j+1}\cup\{s^j\}$.

Now, with inequality (37.19) established, we rewrite it by multiplying both sides by β:

$$\beta \sum_{s \in A^j} \left(w_s - \sum_{e \in s} \gamma_e^{j-1} \right) \geq \beta \sum_{s \in A^{j+1}} \left(w_s - \sum_{e \in s} \gamma_e^j \right) + \beta \Delta^j$$

Next, using the condition of the theorem, that is, $\sum_{e \in \mathcal{E}} \delta_e^j \leq \beta \Delta^j$, the induction hypothesis (37.17) and the fact $\gamma^j - \delta^j = \gamma^{j-1}$, we can bound the last expression from below by

$$\sum_{e \in \mathcal{E}} \left(\gamma_e^p - \gamma_e^j \right) + \sum_{e \in \mathcal{E}} \delta_e^j = \sum_{e \in \mathcal{E}} \left(\gamma_e^p - \gamma_e^{j-1} \right)$$

Thus, we establish inequality (37.18), which proves the theorem. □

The weak duality theorem for linear programming implies $OPT(\mathcal{I}) \leq \sum_{e \in \mathcal{E}} \gamma_e(\mathcal{I})$, where $OPT(\mathcal{I})$ is the optimum value of WSP for instance $\mathcal{I}$ and $\gamma(\mathcal{I})$ is feasible covering γ returned by local covering for $\mathcal{I}$. Let $A(\mathcal{I})$ be the set A returned by the algorithm for $\mathcal{I}$, then from Theorem 37.1 we have

$$OPT(\mathcal{I}) \leq \sum_{e \in \mathcal{E}} \gamma_e(\mathcal{I}) \leq \beta \sum_{s \in A(\mathcal{I})} w_s$$

Corollary 37.1

If (a particular setup of) the local covering algorithm runs in polynomial time and the condition of Theorem 37.1 is satisfied, then the local covering is a $\frac{1}{\beta}$-approximation algorithm.

37.2.2 Algorithm ALG1

Suppose we are given an instance $\mathcal{I}$ of JI with unit capacities. Recall that this is a grid consisting of t columns, numbered consecutively from left to right, and m numbered rows together with a set of intervals $I = \{1, 2, \ldots, n\}$ lying on the rows of the grid. An interval $i \in I$ is specified by the triple (l_i, r_i, ρ_i), where l_i, r_i are the indices of the left- and the rightmost columns intersecting the interval and ρ_i is the index of the row where it lies. For each interval $i \in I$ we are given a positive integral parameter w_i referred to as the interval demand. We assume that the intervals are ordered according to nondecreasing r_i.

Recall that the job interval packing problem (JIP) with unit capacities, can be formulated as follows: *select a maximum-weight subset of intervals A such that no two intervals share a column or row.* Observe that that this problem is a special case of the WSP, considered in the previous section. Indeed, let the ground set $\mathcal{E}$ be the set of all the columns and rows of the grid and the collection $\mathcal{S}$ be $\{s_1, \ldots, s_n\}$, where s_i is the subset of columns and the row stabbing interval i, that is, $s_i = \{column\ l_i, \ldots, column\ r_i, row\ \rho_i\}$. The weights of the subsets are equal to the corresponding interval weights: $w_{s_i} = w_i, \forall i \in I$. It is easy to see that any feasible packing corresponds to a feasible solution to JIP with unit capacities of the same value.

Below we give an ILP formulation of JIP with unit capacities, where we use a characteristic vector x of A as a vector of decision variables:

$$\text{JIP:} \quad \text{Maximize} \quad \sum_{i=1}^{n} w_i x_i \tag{37.20}$$

$$\text{subject to} \quad \sum_{i:\rho_i = r} x_i \leq 1 \quad \forall r = 1, \ldots, m \tag{37.21}$$

$$\sum_{i:c \in [l_i, r_i]} x_i \leq 1 \quad \forall c = 1, \ldots, t \tag{37.22}$$

$$x_i \in \mathbb{Z}_+^1 \qquad \forall i = 1, \ldots, n \tag{37.23}$$

The dual to its LP relaxation (the dual variables $z \in \mathbb{R}^m$ and $y \in \mathbb{R}^t$ correspond to the constraints [37.21] and [37.22] respectively) looks as follows:

$$\text{Dual:} \quad \text{Minimize} \quad \sum_{c=1}^{t} y_c + \sum_{r=1}^{m} z_r \tag{37.24}$$

$$\text{subject to} \quad z_{\rho_i} + \sum_{c \in [l_i, r_i]} y_c \geq w_i \quad \forall i = 1, \ldots, n \tag{37.25}$$

$$z_r, y_c \geq 0 \qquad \forall r, c \tag{37.26}$$

Note that by replacing in this formulation the nonnegativity constraints (37.26) with integrality constraints

$$z_r, y_c \in \mathbb{Z}_+, \ \forall r, c, \tag{37.27}$$

we obtain an ILP formulation of the job interval stabbing JIS problem with unit capacities: *For each column c and row r specify integral multiplicities y_c and z_r, respectively such that*

- *for each interval i the sum of multiplicities of the columns and the row stabbing interval i is at least the demand w_i,*
- *the sum of the multiplicities is minimum.*

Note that in the case of JIS with unit capacities the interval multiplicities can be fixed to zero without loss in the optimum value, since, given any feasible solution, one can obtain a feasible solution of the same value by decreasing interval multiplicities to zero and increasing row multiplicities so as to preserve the feasibility.

We now describe a setup of the generic algorithm Local Covering for JIP with unit capacities, yielding a primal-dual approximation algorithm called ALG1.

Let us reproduce step 4 of local covering (see Figure 37.2) and translate it into the terms of JIP with unit capacities.

Line (1), that is, selecting a violated subset $s^l = s_i$ corresponds in our context to selecting interval i such that $\sum_{c \in [l_i, r_i]} y_c + z_{\rho_i} < w_i$. When more then one index i satisfies this condition we select the smallest of them.

Line (2) should be translated as $\Delta \leftarrow w_i - \sum_{c \in [l_i, r_i]} y_c - z_{\rho_i}$.

In line (3) we have to find $\delta = (\delta_{col1}, \ldots, \delta_{colt}, \ \delta_{row1}, \ldots, \delta_{rowm})$ s.t.

$$\sum_{e \in s_j} \delta_e \geq \min\left(w_j - \sum_{c \in [l_j, r_j]} y_c - z_{\rho_j}, \ \Delta \right), \quad \forall j \text{ s.t. } s_j \in N(s_i) \tag{37.28}$$

where i is the number selected in line (1). We assign the value of Δ to the elements of δ corresponding to the right-most column stabbing interval i and to its row, and 0 to all the other elements.

```
Until $\gamma$ is a feasible covering
  $l \leftarrow l + 1$
(1) Select $s^l$ such that $\sum_{e \in s^l} \gamma_e < w_{s^l}$
(2) $\Delta \leftarrow w_{s^l} - \sum_{e \in s^l} \gamma_e$
(3) Find $\delta \in \mathbb{R}_+^{|\mathcal{E}|}$ s. t.: $\sum_{e \in s} \delta_e \geq \min(w_s - \sum_{e \in s} \gamma_e, \Delta), \forall s \in N(s^l)$
(4) $\gamma \leftarrow \gamma + \delta$
```

FIGURE 37.2 Step 4 of Local Covering.

```
1. $y \leftarrow \mathbf{0},\ z \leftarrow \mathbf{0}$
2. $A \leftarrow \emptyset$
3. $l \leftarrow 0$
4. For $i \leftarrow 1$ to $n$
     $\Delta \leftarrow w_i - \sum_{c\in[l_i,\, r_i]} y_c - z_{\rho_i}$,
     if $\Delta > 0$ then:
       $l \leftarrow l + 1,\ y_{r_i} \leftarrow y_{r_i} + \Delta,\ z_{\rho_i} \leftarrow z_{\rho_i} + \Delta,\ i_l \leftarrow i$
5. For $l$ downto 1
     if $A \cup \{i_l\}$ is feasible to $JIP$ then $A \leftarrow A \cup \{i_l\}$
6. Return $A$ (and $y$, $z$)
```

FIGURE 37.3 Algorithm ALG1.

Lemma 37.2

If vectors z and y and index i are such that $w_j - \sum_{c\in[l_j,r_j]} y_c - z_{\rho_j} \le 0$ *for all* $j = 1, \ldots, i-1$, *then vector* δ *defined as*

$$\delta_e = \begin{cases} \Delta, & \textit{if } e = \textit{col}\, r_i \\ \Delta, & \textit{if } e = \textit{row}\, \rho_i \\ 0, & \textit{otherwise} \end{cases}$$

satisfies (37.28) for any $\Delta > 0$.

Proof

The condition of the lemma guarantees that Eq. (37.28) is satisfied for all $s_j \in N(s_i)$ such that $j < i$.

Consider the other neighbors of s_i, that is, all $s_j \in N(s_i)$ such that $j \ge i$. These subsets correspond to the intervals that either lie on the same row or share a column with interval i and whose right-most stabbing column has index at least equal to r_i. Then sharing a column with interval i implies sharing the column r_i. This means that each subset $s_j \in N(s_i),\ j \ge i$, includes either row ρ_i or column r_i. Thus,

$$\forall s_j \in N(s_i),\ \text{s.t. } j \ge i:\ \sum_{e\in s_j} \delta_e \ge \min(\delta_{col\ r_i},\ \delta_{row\ \rho_i}) = \Delta$$

This implies the lemma. □

Figure 37.3 shows the formal description the algorithm ALG1 (see also Ref. [12]). Here A is a set of interval indices.

Theorem 37.2

For any instance $\mathcal{I}$ *of JI with unit capacities, the set A and the vectors* (y, z) *returned by the algorithm ALG1 describe feasible solutions to JIP and JIS respectively, and their values (denoted by val*(y, z) *and val*(A)*, respectively) are related as*

$$\textit{val}(y, z) \le 2 \cdot \textit{val}(A)$$

Proof

Obviously A is a feasible solution to JIP with unit capacities. Consider (y, z). According to Theorem 37.1 it is a feasible covering, that is, it is a feasible solution to the LP formulation (37.24)–(37.26). Obviously the vectors y and z are integral since w_i is integral for all $i \in I$. Thus (y, z) is a feasible solution to the ILP formulation (37.24),(37.25),(37.27).

Let us establish the ratio of 2 between the values of the solutions. Observe that the conditions of Theorem 37.1 are satisfied with $\beta = 2$. Indeed, at each iteration of step 4 of the algorithm $\sum_{e \in \mathcal{E}} \delta_e = 2\Delta$. Theorem 37.1 implies that

$$\sum_{r=1}^{m} z_r + \sum_{c=1}^{t} y_c \leq 2 \cdot \sum_{i \in A} w_i \qquad \square$$

Algorithm ALG1 can be implemented to run in $O(n \log n)$ time [3].

From the weak duality relation between JIP and JIS, and from Theorem 37.2, we have that for any instance $\mathcal{I}$ of JI with unit capacities

$$JIP(\mathcal{I}) \leq \mathrm{val}(y(\mathcal{I}), z(\mathcal{I})) \leq 2 \cdot \mathrm{val}(A(\mathcal{I})) \leq 2JIS(\mathcal{I})$$

where $JIP(\mathcal{I})$ and $JIS(\mathcal{I})$ are the optimal values of JIP and JIS for $\mathcal{I}$, and $(y(\mathcal{I}), z(\mathcal{I}))$, and $A(\mathcal{I})$ are the values of the solutions returned by algorithm ALG1 applied to $\mathcal{I}$.

Corollary 37.2

Algorithm ALG1 is a 1/2-approximation algorithm for JIP with unit capacities and 2-approximation for JIS with unit capacities.

The results stated in Corollary 37.2 are tight, that is, the analysis of the algorithm's performance cannot be improved to provide a better factor [12].

37.3 The Case of Unit Demands: Algorithm ALG2

In this section we focus on JI with unit demands. This special case of JI can be described as follows: given is a grid consisting of t columns, numbered consecutively from left to right, and m numbered rows together with a set of intervals $I = \{1, 2, \ldots, n\}$ lying on the rows of the grid. An interval i is specified by the triple (l_i, r_i, ρ_i), where l_i, r_i are the indices of the leftmost and the rightmost columns intersecting the interval and ρ_i is the index of the row where it lies. For each column c, row r, and interval i, we are given positive integral parameters v_c, u_r, and p_i respectively, referred to as the column, row, and interval capacities. We assume that the intervals are ordered according to nondecreasing r_i.

The objective of the JIS with unit demands is: *find a collection H of columns, rows, and intervals of minimum total capacity, such that for each interval $i \in I$, H contains either a column stabbing interval i, or the row stabbing interval i, or the interval i itself.*

Note that JIS with unit demands is a special case of the well-known *weighted hitting set* problem (WHS): *given is a finite weighted ground set $\mathcal{E}$, each element $e \in \mathcal{E}$ having a nonnegative weight w_e, and a collection of its subsets $\mathcal{S}$. Find a minimum-weight subset $H \subseteq \mathcal{E}$ such that $H \cap s \neq \emptyset$ for any $s \in \mathcal{S}$.*

Indeed, let the set of all the columns, rows, and intervals with their weights constitute a weighted ground set $\mathcal{E}$. Let $\mathcal{S}$ be $\{s_1, s_2, \ldots, s_n\}$, where subset s_i contains the interval i together with the subset of the columns and the row stabbing it. Then any feasible hitting set corresponds to a feasible solution to JIS with unit demands of the same value.

In the spirit of WHS we say that an interval i is *hit* by a subset H of columns, rows, and intervals if H contains either column or row stabbing interval i, or interval i itself.

Below we give an ILP formulation of JIS with unit demands (for the decision variables we use here characteristic vector (y, z, s) of H, where $y \in \mathbb{Z}^t$ is associated with the set of columns, $z \in \mathbb{Z}^m$ with the set of rows and $s \in \mathbb{Z}^n$ with the set of intervals).

$$\text{JIS:} \quad \text{Minimize} \quad \sum_{c=1}^{t} v_c y_c + \sum_{r=1}^{m} u_r z_r + \sum_{i=1}^{n} p_i s_i \tag{37.29}$$

$$\text{subject to} \quad z_{\rho_i} + \sum_{c \in [l_i, r_i]} y_c + s_i \geq 1 \quad \forall i \tag{37.30}$$

$$z_r, y_c, s_i \in \{0, 1\} \qquad \forall r, c, i \tag{37.31}$$

The dual to its LP relaxation is

$$\text{Dual:}\quad \text{Maximize} \quad \sum_{i=1}^{n} x_i \tag{37.32}$$

$$\text{subject to} \quad \sum_{i:\rho_i=r} x_i \le u_r \qquad \forall r = 1,\ldots,m \tag{37.33}$$

$$\sum_{i:c\in[l_i,r_i]} x_i \le v_c \qquad \forall c = 1,\ldots,t \tag{37.34}$$

$$x_i \le p_i \qquad \forall i = 1,\ldots,n \tag{37.35}$$

$$x_i \ge 0 \qquad \forall i = 1,\ldots,n \tag{37.36}$$

Note that when replacing the nonnegativity constraints (37.36) with integrality constraints $x_i \in \mathbb{Z}\ \forall i = 1,\ldots,n$, we obtain an ILP formulation of the JIP with unit demands.

Specify an integral multiplicity for each interval i, not exceeding its capacities p_i, such that

- *for each column c or row r the sum of the multiplicities of the intervals sharing it does not exceed the capacity v_c or u_r respectively,*
- *the sum of the multiplicities is minimum.*

Following the framework of the generic *primal-dual algorithm with reverse delete step* described for WHS by Goemans and Williamson [17], we develop a primal-dual algorithm for JIS with unit demands, called ALG2.

We use auxiliary variables $\hat{v} \in \mathbb{R}^t$, $\hat{u} \in \mathbb{R}^m$, and $\hat{p} \in \mathbb{R}^n$ which are in fact slack variables for the constraints (37.33)–(37.35), that is, at each moment in time:

$$\hat{v}_c = v_c - \sum_{i:c\in[l_i,r_i]} x_i \ \forall c, \qquad \hat{u}_r = u_r - \sum_{i:\rho_i=r} x_i \ \forall r, \qquad \hat{p}_i = p_i - x_i \ \forall i \tag{37.37}$$

for some current values of $x_i,\ i = 1,\ldots,n$.

Initially, the dual vector x is zero, the set H is empty and the slack variables $\hat{v}$, $\hat{u}$, $\hat{p}$ are equal to v, u, and p respectively. For each interval $i \in I$ we check whether it is already hit by H, if not we do the following:

- assign to the dual variable x_i the minimum of the values of slack variables corresponding to the following elements: the columns stabbing interval i, its row and interval i itself, that is, $x_i \leftarrow \min\{\hat{v}_{l_i},\ldots,\hat{v}_{r_i},\hat{u}_{\rho_i},\hat{p}_i\}$;
- update the slack variables (since the value of x_i changed). Because of the way x_i was updated, at least one of the slack variables $\hat{v}_{l_i},\ldots,\hat{v}_{r_i},\hat{u}_{\rho_i},\hat{p}_i$ has to be zero now;
- add to H the elements (columns $l_i,\ldots,r_i$, row ρ_i, or interval i) whose corresponding slack variables are zero. Note that at least one of these element has to be added to H and thus interval i becomes hit.

Clearly, after all the n intervals are processed as above, H is a feasible solution to JIS with unit demands, since all the intervals are hit. At the next stage we try to remove elements from H. For that we consider each element in H and remove it if feasibility of H is preserved. The order of considerations plays a role here. First we check the columns in the order reverse to the order they were added to H. Then all the other elements, that is, rows and intervals, in an arbitrary order.

The formal description of algorithm ALG2 is shown in Figure 37.4. We use three index sets to represent set H, the set of column, row, and interval indices H^{col}, H^{row}, and H^{int}, respectively.

```
1. $x \leftarrow \mathbf{0}$
2. $H^{col} \leftarrow \emptyset$, $H^{row} \leftarrow \emptyset$, $H^{int} \leftarrow \emptyset$
3. $\hat{v} \leftarrow v$, $\hat{u} \leftarrow u$, $\hat{p} \leftarrow p$
4. $l \leftarrow 0$
5. For $i \leftarrow 1$ to $n$
     If $([l_i, r_i] \cap H^{col} = \emptyset)$ $AND$ $(\{\rho_i\} \cap H^{row} = \emptyset)$ then
       $x_i \leftarrow \min\{\hat{v}_{l_i}, ..., \hat{v}_{r_i}, \hat{u}_{\rho_i}, \hat{p}_i\}$;
       For $c \leftarrow l_i$ to $r_i$ if $(\hat{v}_c \leftarrow \hat{v}_c - x_i) = 0$ then
                                 $l \leftarrow l + 1$, $c_l \leftarrow c$, $H^{col} \leftarrow H^{col} \cup \{c\}$
       If $(\hat{u}_{\rho_i} \leftarrow \hat{u}_{\rho_i} - x_i) = 0$ then $H^{row} \leftarrow H^{row} \cup \{\rho_i\}$
       If $(\hat{p}_i \leftarrow \hat{p}_i - x_i) = 0$ then $H^{int} \leftarrow H^{int} \cup \{i\}$
6. For $j \leftarrow l$ downto 1
     if $H^{col} - \{c_j\}$, together with $H^{row}$ and $H^{int}$, is feasible
       then $H^{col} \leftarrow H^{col} - \{c_j\}$
7. For all $i \in H^{int}$
     if $H^{int} - \{i\}$ together with $H^{col}$ and $H^{row}$ is feasible
       then $H^{row} \leftarrow H^{row} - \{i\}$
8. For all $r \in H^{row}$
     if $H^{row} - \{r\}$, together with $H^{col}$ and $H^{int}$, is feasible
       then $H^{row} \leftarrow H^{row} - \{r\}$
9. Return $H^{col}$, $H^{row}$, $H^{int}$ (and $x$)
```

FIGURE 37.4 Algorithm ALG2.

Theorem 37.3

For any instance $\mathcal{I}$ of JI with unit demands the sets $(H^{col}, H^{row}, H^{int})$ and vector x returned by the algorithm ALG2 describe feasible solutions to JIS and JIP respectively, and their values are related as follows:

$$val(H^{col}, H^{row}, H^{int}) \leq 2\ val(x)$$

To prove this we need a preliminary lemma. This is a result for the WHS that can be also found in Ref. [17]:

Lemma 37.3

Consider an instance of WHS. If a set $H \subset \mathcal{E}$, vector $\chi \in \mathbb{R}^{|\mathcal{S}|}$ and $\beta \geq 0$ satisfy

$$\forall e \in H: \sum_{s \ni e} \chi_s = w_e \quad and \quad \forall s \in \mathcal{S},\ such\ that\ \chi_s > 0: |s \cap H| \leq \beta$$

then

$$\sum_{e \in H} w_e \leq \beta \sum_{s \in \mathcal{S}} \chi_s$$

Proof

Using the conditions of the lemma we have

$$\sum_{e \in H} w_e = \sum_{e \in H} \sum_{s \ni e} \chi_s = \sum_{s \in \mathcal{S}} |s \cap H| \chi_s \leq \beta \sum_{s \in \mathcal{S}} \chi_s \qquad \square$$

Proof (of the Theorem)
Obviously, by construction the sets $(H^{col}, H^{row}, H^{int})$ describe a feasible solution to JIS with unit demands and the vector x is feasible to the dual LP formulation (37.33)–(37.35). Note, that the integrality of the input data implies integrality of x. Thus x is feasible to JIP with unit demands.

Let us establish the relation between the solution values. Recall the representation of JIS with unit demands as a special case of WHS, described earlier in this section. We use the result of the lemma with $(H^{col}, H^{row}, H^{int})$ representing H, x representing χ, β equal to 2.

Let us show that the first condition of the lemma is satisfied. Recall that an index of an element (which can be a column, row, or interval) is added to one of $(H^{col}, H^{row}, H^{int})$ only when the corresponding slack variable (37.37) becomes zero. After a slack variable becomes zero, it is not changed by the algorithm anymore. Thus at the end of the algorithm we have all the slack variables (37.37) corresponding to the elements in the solution to be zero, and thus the first condition of the lemma follows.

Let us establish the second condition, that is, show that for $(H^{col}, H^{row}, H^{int})$ returned by the algorithm and for any $i \in I$, such that $x_i > 0$ holds:

$$|\{l_i, \ldots, r_i\} \cap H^{col}| + |\{\rho_i\} \cap H^{row}| + |\{i\} \cap H^{int}| \leq 2$$

Take i such that $x_i > 0$. If we show that $|\{l_i, \ldots, r_i\} \cap H^{col}| \leq 1$, the above bound follows easily from the fact that, due to the minimality of solution $(H^{col}, H^{row}, H^{int})$, accomplished in steps 6–8. $|\{i\} \cap H^{int}| = 0$ for any i, for which $|\{l_i, \ldots, r_i\} \cap H^{col}| + |\{\rho_i\} \cap H^{row}| \geq 1$.

Suppose $|\{l_i, \ldots, r_i\} \cap H^{col}| \geq 2$, that is, H^{col} contains at least two columns incident to the interval i, say, columns c_1 and c_2, $c_1 < c_2$. Consider the moment right before x_i became positive, that is, the beginning of the ith iteration at step 5 of the algorithm. All the previously considered intervals j, $j < i$, are already covered by this moment and neither c_1, nor c_2 are yet added to H^{col} (otherwise, $\{l_i, l_i + 1, \ldots, r_i\} \cap H^{col} = \emptyset$ would not hold). Look now at step 6, the moment when we are considering column c_1 as a candidate for removal from H^{col}. We claim that all intervals j, $j < i$, are currently covered by other elements (columns, rows, or intervals). This is because of the fact that no element, added to the solution during step 5 before column c_1, can be considered for removal in step 6 before c_1. Further, it is not difficult to see that all intervals l, $l \geq i$, stabbed by column c_1, have to be stabbed by column c_2 as well by the ordering of the intervals (see Figure 37.5). Therefore, nothing can prevent us from removing c_1 from H^{col} in step 6. □

Algorithm ALG2 can be implemented to run in $O(nt)$ time [3].

Theorem 37.3 together with the weak duality relation between JIP and JIS has the following consequence.

Corollary 37.3

Algorithm ALG2 is a 1/2-approximation algorithm for JIP with unit demands and a 2-approximation for JIS with unit demands.

Again, these performance guarantees are tight (see Kovaleva and Spieksma [12]). Finally, note that both algorithms ALG1 and ALG2 can be applied to instances of JI with unit capacities and demands. It is not difficult to verify that the solutions for JIP (subsets of intervals) returned by the two algorithms coincide, while the solutions to JIS (subsets of columns and rows) may be different.

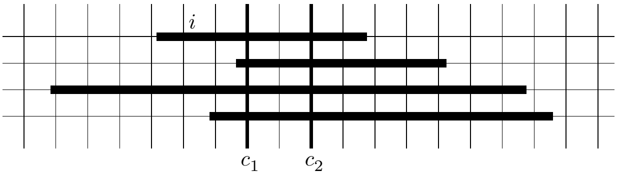


FIGURE 37.5 All the intervals l, $l \geq i$, incident to column c_1 have to be incident to column c_2 as well.

References

[1] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M., *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties*, Springer, Berlin, 1999.

[2] Vazirani, V. V., *Approximation Algorithms*, Springer, Berlin, 2001.

[3] Kovaleva, S., Approximation of Geometric Set Packing and Hitting Set Problems, Ph.D. thesis, Maastricht University, 2003.

[4] Berman, P. and DasGupta, B., Multi-phase algorithms for throughput maximization for real-time scheduling, *J. Comb. Opt.*, 4, 307, 2000.

[5] Erlebach, T. and Spieksma, F. C. R., Interval selection: applications, algorithms, and lower bounds, *J. Algorithms*, 46, 27, 2003.

[6] Bar-Noy, A., Guha, S., Naor, J., and Schieber, B., Approximating the throughput of multiple machines in real-time scheduling, *SIAM J. Comput.*, 31, 331, 2001.

[7] Spieksma, F. C. R., On the approximability of an interval scheduling problem, *J. Scheduling*, 2, 215, 1999.

[8] Chuzhoy, J., Ostrovsky, R., and Rabani, Y., Approximation algorithms for the job interval selection problem and related scheduling problems, *Proc. FOCS,* 2001.

[9] Calinescu, G., Chakrabarti, A., Karloff, H., and Rabani, Y., Improved approximation algorithms for resource allocation, *Proc. IPCO Conf.*, Lecture Notes in Computer Science, Vol. 2337, Springer, Berlin, 2002, p. 401.

[10] Bar-Noy, A., Bar-Yehuda, R, Freund, A., Naor, J., and Schieber, B., A unified approach to approximating resource allocation and scheduling, *JACM*, 48, 1069, 2001.

[11] Hassin, R. and Megiddo, N., Approximation algorithms for hitting objects with straight lines, *Disc. Appl. Math.*, 30, 29, 1991.

[12] Kovaleva, S. and Spieksma, F. C. R., Primal-dual approximation algorithms for a packing-covering pair of problems, *RAIRO-Oper. Res.*, 36, 53, 2002.

[13] Kovaleva, S. and Spieksma, F. C. R., Approximation algorithms for rectangle stabbing and interval stabbing problems, *SIAM J. Disc. Math.*, 20, 748, 2006.

[14] Gaur, D., Ibaraki, T., and Krishnamurti, R., Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem, *J. Algorithms*, 43, 138, 2002.

[15] Akcoglu, K., Aspres, J., DasGupta, B., and Kao, M.-Y., Opportunity cost algorithms for combinatorial auctions, in *Applied Optimization: Computational Methods in Decision-Making, Economics, and Finance*, Kontoghiorghes, E. J., Rustem, B., and Siokos, S., Eds., *Applied Optimization: Computational Methods in Decision-Making, Economics, and Finance*, Kluwer Academic, Dordrecht, 2002.

[16] Bar-Yehuda, R. and Even, S., A local-ratio theorem for approximating the weighted set cover problem, *Ann. Disc. Math.*, 25, 27, 1985.

[17] Goemans, M. X. and Williamson, D. P., The primal-dual method for approximation algorithms and its application to network design problems, in *Approximation Algorithms for NP-Hard Problems*, Hochbaum, D. S., Ed., PWC Publishing Company, Boston, 1996.

38

Approximation Algorithms for Facility Dispersion

S. S. Ravi*
University at Albany—State University of New York

Daniel J. Rosenkrantz†
University at Albany—State University of New York

Giri K. Tayi
University at Albany—State University of New York

38.1 Introduction

Many problems in location theory deal with the placement of facilities on a network so as to minimize some function of the distances between facilities or between facilities and other nodes of the network [1]. Such problems model the placement of "desirable" facilities such as warehouses, hospitals, and fire stations. However, there are situations in which facilities must be located so as to *maximize* a given function of the distances. Such location problems are referred to as **dispersion** problems [2] since they model situations in which proximity of facilities is undesirable. One example of such a situation involves placing "obnoxious" (also called "undesirable") facilities such as nuclear power plants, oil storage tanks, and ammunition dumps [2,3]. These facilities need to be spread out to the greatest possible extent so that an accident at one of the facilities does not damage any of the others. Another example where dispersion problems arise is in the distribution of business franchises in a city [2,4]. In this case, separation of business units is desirable to minimize the competition for customers among the units. In these examples, the facilities to be dispersed are assumed to be of the same type. Applications involving multiple types of facilities (e.g., incinerators and landfills) have also been considered in the literature [3].

The concept of dispersion is also useful in the context of multiobjective decision making [5]. When the number of nondominated[1] solutions is large, a decision maker may be interested in selecting a manageable collection of solutions which are dispersed as far as possible with respect to the various

*Supported by NSF Grant CCR-97-34936.

†Supported by NSF Grant CCR-01-05536.

[1] Given two solutions S_1 and S_2 to a multiobjective optimization problem, S_1 **dominates** S_2 if S_1 is no worse than S_2 in every objective and S_1 is strictly better than S_2 in at least one objective. It can be seen that the domination relation is a partial order. Each maximal element of this partial is a **nondominated** solution.

objectives. Identifying such a collection is useful in obtaining an understanding of the range of available alternatives.

Dispersion problems can be formulated under a variety of maximization objectives. Many of these optimization problems are **NP**-hard. So, it is unlikely that optimal solutions to these problems can be obtained efficiently. The practical importance of these problems motivates the study of efficient approximation algorithms that provide near-optimal solutions. In this chapter, we present some approximation algorithms for several versions of the dispersion problem. This chapter is not a survey on FD, rather, the focus is on approximation algorithms for which performance guarantee results have been established.

The remainder of this chapter is organized as follows. Section 38.2 provides a general formulation of the dispersion problem and mentions a number of objectives considered in the literature. Section 38.3 presents approximation algorithms for the dispersion problem under several objectives. The discussion also includes a greedy approach that is useful in designing such approximation algorithms. Section 38.4 presents approximation algorithms for capacitated versions of dispersion problems. Section 38.5 discusses several directions for future research.

38.2 Analytical Model and Problem Formulation

38.2.1 Formulation of Dispersion Problems

Analytical models for the dispersion problem assume that the input consists of a set $V = \{v_1, v_2, \ldots, v_n\}$ of n nodes, a nonnegative distance (also called **edge weight**) between each pair of nodes in V and an integer $p \leq n$. Distances are assumed to be symmetric so that the input can be thought of as an undirected complete graph on n nodes with a nonnegative weight on each edge. The weight of the edge $\{v_i, v_j\}$ $(i \neq j)$ is denoted by $w(v_i, v_j)$. It is assumed that $w(v_i, v_i) = 0$ for $1 \leq i \leq n$. Under these assumptions, a general formulation of the FD problem is as follows.

Facility Dispersion
Instance
A complete graph $G(V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$, a nonnegative distance $w(v_i, v_j)$ for each edge $\{v_i, v_j\}$ in E, and an integer $p \leq n$.

Requirement
Find a subset $P \subseteq V$, with $|P| = p$, such that a specified objective function $f(P)$ is maximized.

To avoid trivial solutions, we will assume throughout this chapter that $p \geq 2$. The chosen placement P induces a complete subgraph of G, denoted by $G(P)$. The objective $f(P)$ is a function of the edge distances in $G(P)$. A number of alternatives for $f(P)$ have been considered in the literature (see, Refs. [3,6–14] and the references cited therein). Some examples of such objective functions are listed below.

(a) *Max-Min dispersion.* Here, the function $f(P)$ is defined to be the smallest edge weight in $G(P)$. Thus, the goal of the corresponding dispersion problem is to maximize the smallest edge weight in $G(P)$. (This is referred to as the **remote edge** problem in Ref. [14].)

(b) *Max-average dispersion.* Here, the function $f(P)$ is defined to be the average edge weight in $G(P)$. Since the number of edges in $G(P)$ is $p(p-1)/2$, $f(P)$ is given by

$$f(P) = \frac{2\sum_{\{v_i,v_j\}\in G(P)} w(v_i, v_j)}{p(p-1)} \tag{38.1}$$

As the denominator $p(p-1)$ is independent of which nodes are in P, maximizing the value of $f(P)$ given by Eq. (38.1) is equivalent to maximizing the sum of the edge weights in $G(P)$. (This is referred to as the **remote clique** problem in Ref. [14].)

(c) *Max-MST dispersion.* Here, the function $f(P)$ is defined to be the weight of a minimum spanning tree (MST) of $G(P)$. A variant of this is the Max-ST dispersion measure, where $f(P)$ is the weight of a minimum Steiner tree [15] (ST) of $G(P)$.

(d) *Max-TSP dispersion.* Here, the function $f(P)$ is defined to be the weight of a minimum traveling salesperson (TSP) tour of $G(P)$; that is, $f(P)$ is the minimum weight of a simple cycle containing all the nodes in P.

(e) *Max-matching dispersion.* Here, $f(P)$ is the weight of a minimum perfect matching[2] in $G(P)$.

(f) *Max-star dispersion.* Here, the dispersion objective $f(P)$ is given by

$$f(P) = \min_{v_i \in P} \left\{ \sum_{v_j \in P} w(v_i, v_j) \right\} \tag{38.2}$$

(This problem is referred to as the **remote star** problem in Ref. [14].)

Under all of the above objectives, the dispersion problem is known to be **NP**-hard. This motivates the study of approximation algorithms for such problems.

The edge distances specified in an instance of the dispersion problem are said to satisfy the **triangle inequality** when for any three distinct nodes x, y, and z, $w(x, z) \leq w(x, y) + w(y, z)$. We refer to such instances as **metric** instances. Most of the approximation algorithms reported in the literature are for metric instances of dispersion problems. Nonapproximability results are known for some nonmetric instances of dispersion problems [12–14].

Geometric versions of dispersion problems have been considered in the literature under two models, which we refer to as the **discrete placement** and **continuous placement** models respectively. Under the discrete placement model [11–13,16,17], the node set V consists of n points in an appropriate metric space (e.g., Euclidean space) along with a chosen distance function (e.g., L_k distance metric for some $k \geq 1$), and the placement must be a subset of the points in V. Under the continuous placement model, a region in which facilities must be placed is specified geometrically (e.g., a polygonal region possibly containing holes), and each facility may be placed at any point inside or on the boundary of the specified region [18]. In discussing geometric versions, we use the terms "point" and "node" interchangeably.

38.2.2 Additional Definitions

This section contains several definitions that are used throughout this chapter. Definitions of common graph theoretic and algorithmic notions used in this chapter can be found in standard texts such as in Refs.[19,20].

For a maximization problem, a polynomial-time approximation algorithm provides a **performance guarantee** of $\rho \geq 1$, if for each instance of the problem, the solution value produced by the algorithm is at least $1/\rho$ of the optimal solution value. We will also refer to such an algorithm as a ρ**-approximation algorithm.** A **polynomial-time approximation scheme** (PTAS) for a problem is a family of polynomial-time algorithms such that for each fixed $\epsilon > 0$, the family contains an algorithm that provides a performance guarantee of $(1 + \epsilon)$.

A **matching** in an undirected graph $G(V, E)$ is a subset E' of edges such that no two edges in E' are incident on the same node. A **maximum matching** is a matching containing the largest number of edges. When the edges of G have weights, a **maximum-weight matching** is a matching with the largest total weight. For each $k \geq 1$, a k**-matching** is a matching consisting of exactly k edges. When the edges of G have weights, a **maximum-weight** k**-matching** (**minimum-weight** k**-matching**) is a k-matching with the largest (smallest) total weight. A graph $G(V, E)$ has a **perfect matching** if it has a matching of size $\lfloor |V|/2 \rfloor$. When the edges of G have weights, a **maximum-weight perfect matching** (**minimum-weight perfect matching**) is a perfect matching with the largest (smallest) total weight.

A subset V' of nodes of an undirected graph $G(V, E)$ is an **independent set** if there is no edge between any pair of nodes in V'. Given an undirected graph $G(V, E)$ and an integer $J \leq |V|$, the goal of the Maximum Independent Set (MIS) decision problem is to determine whether G has an independent set

[2] See Section 38.2.2 for the definition of perfect matching.

of size at least J. This decision problem is known to be **NP**-complete [15]. When there are nonnegative weights on nodes, one can generalize the problem of finding a MIS to the problem of finding an independent set of maximum weight. Strong nonapproximability results are known for the MIS (optimization) problem for general graphs. In particular, it has been shown [21] that for any $\epsilon > 0$, the MIS problem cannot be approximated to within a factor $O(n^{1-\epsilon})$, unless the complexity classes[3] **NP** and **ZPP** coincide.

In developing approximation algorithms for capacitated dispersion problems (Section 38.4), the class of **unit disk graphs** plays an important role. An undirected graph is a **unit disk graph** if its vertices can be placed in one-to-one correspondence with a set of circles of equal radius in the plane so that two vertices of the graph are joined by an edge if and only if the corresponding circles touch or intersect [23]. The geometric representation for a unit disk graph consists of the radius value and the coordinates of the center of each disk. The recognition problem for unit disk graphs is **NP**-hard [24]. When the geometric representation is not available, the MIS problem for unit disk graphs has a constant factor approximation [25]. However, given the geometric representation, there is a PTAS for the MIS problem for unit disk graphs [26]. As mentioned in Ref. [27], this PTAS can be extended in a straightforward manner to the weighted MIS problem for unit disk graphs.

38.3 Approximating Dispersion Problems

38.3.1 Overview

In this section, we discuss approximation algorithms with good performance guarantees for dispersion problems under several objectives. We begin with a greedy approach for designing an approximation algorithm for metric instances of the Max-Min dispersion problem. We observe that the greedy approach can also be used to obtain approximation algorithms for other dispersion objectives. Further, we point out that for some objectives, approximation algorithms developed from the greedy framework can be improved using other techniques. Known results for geometric instances of problems are also summarized.

38.3.2 Approximation Algorithms for Max-Min Dispersion

38.3.2.1 Results for Metric Instances

We abbreviate the Max-Min facility dispersion problem by MMFD. It is shown in Ref. [12] that for any $\rho \geq 1$, if there is a polynomial-time ρ-approximation algorithm for nonmetric instances of the MMFD problem, then the MIS problem can be solved in polynomial-time. In other words, for any $\rho \geq 1$, there is no ρ-approximation algorithm for nonmetric instances of MMFD, unless **P** = **NP**. Therefore, in this section, we restrict our attention to metric instances of MMFD, denoted by MMFD:TI. (The suffix "TI" is used to indicate that the distances satisfy the triangle inequality.)

A greedy heuristic for MMFD:TI, called GMM, is shown in Figure 38.1. This heuristic is similar to "furthest point outside the neighborhood" heuristic, described in Ref. [5]. For results regarding the performance guarantee provided by this heuristic see Refs. [10–12,28]. The presentation below is based on Ref. [12].

The heuristic begins by initializing P (the set of nodes at which facilities are placed) to contain a pair of nodes in V which are joined by an edge of maximum weight. Subsequently, each iteration of GMM chooses a node v from $V - P$ such that the minimum distance from v to a node in P is the largest among all the nodes in $V - P$. In each step, ties are broken arbitrarily. Heuristic GMM terminates when $|P| = p$. The solution value of the placement P produced by GMM is equal to $\min_{x, y \in P}\{w(x, y)\}$. Using standard techniques, it can be seen that the running time of the heuristic is $O(n^2)$.

Our next theorem shows that GMM provides a performance guarantee of 2. It will also be shown (Theorem 38.2) that unless **P** = **NP**, no polynomial-time heuristic can provide a better performance guarantee.

[3] For definitions of complexity classes such as **ZPP**, see [22].

Input: A complete graph $G(V, E)$ with a nonnegative edge-weight $w(v_i, v_j)$ for each edge $\{v_i, v_j\} \in E$ and an integer $p \leq |V|$. (The edge weights satisfy the triangle inequality.)

Output: A set $P \subseteq V$ such that $|P| = p$. (The goal is to make the smallest edge weight in $G(P)$ close to the optimum value.)

Algorithm:

```
1. Let $P = \emptyset$.
2. Let $v_i$ and $v_j$ be the endpoints of an edge of maximum weight. Add the nodes $v_i$ and $v_j$ to $P$.
3. while ($|P| < p$) do
    (a) Find a node $v \in V - P$ such that $\min_{v' \in P} \{w(v, v')\}$ is maximum among the nodes in $V - P$.
    (b) Add $v$ to $P$.
4. Output $P$.
```

FIGURE 38.1 Details of heuristic GMM.

Theorem 38.1

Let I be an instance of MMFD:TI. Let OPT(I) and GMM(I) denote respectively the solution values of an optimal placement and that produced by GMM for the instance I. Then $OPT(I)/GMM(I) \leq 2$.

Proof

Consider the set-valued variable P in the description of heuristic GMM. Let $f(P) = \min_{x,y \in P}\{w(x, y)\}$. We will show by induction that the condition

$$f(P) \geq \text{OPT}(I)/2 \tag{38.3}$$

holds after each addition to P. Since $\text{GMM}(I) = f(P)$ after the last addition to P, the theorem would follow.

Since the first addition inserts two nodes joined by an edge of largest weight into P, Condition (38.3) clearly holds after the first addition. So, assume that the condition holds after k additions to P, for some k, $1 \leq k < p - 1$. We will prove that the condition holds after the $(k + 1)$th addition to P as well.

To that end, let $P^* = \{v_1^*, v_2^*, \ldots, v_p^*\}$ denote an optimal placement. For convenience, we use ℓ^* for OPT(I). The following observation is an immediate consequence of the definition of the solution value corresponding to a placement for an MMFD instance.

Observation 38.1

For every pair v_i^*, v_j^* of distinct nodes in P^*, $w(v_i^*, v_j^*) \geq \ell^*$. ■

Let $P_k = \{x_1, x_2, \ldots, x_{k+1}\}$ denote the set P after k additions, $1 \leq k \leq p - 1$. (Note that $|P_k| = k + 1$, since the first addition inserts two nodes into P). Since GMM adds at least one more node to P, the following is a trivial observation.

Observation 38.2

For $1 \leq k < p-1, |P_k| = k+1 < p$. ■

For each $v_i^* \in P^*$ $(1 \leq i \leq p)$, define $S_i^* = \{u \in V \mid w(v_i^*, u) < \ell^*/2\}$. The following claim provides two useful properties of these sets.

Claim 38.1

(a) For $1 \leq i \leq p$, S_i^* *is nonempty.*
(b) For $i \neq j$, S_i^* *and* S_j^* *are disjoint.*

Proof

Part (a) is obvious, since $v_i^* \in S_i^*$ for $1 \leq i \leq p$. To prove part (b), suppose $S_i^* \cap S_j^* \neq \emptyset$ for some $i \neq j$. Let $u \in S_i^* \cap S_j^*$. Thus, $w(v_i^*, u) < \ell^*/2$ and $w(v_j^*, u) < \ell^*/2$. By Observation 38.1, $w(v_i^*, v_j^*) \geq \ell^*$. These three inequalities together imply that the triangle inequality does not hold for the three nodes u, v_i^*, and v_j^*. Part (b) follows. ■

We now continue with the main proof. Since for $k < p-1$, P_k has *less than* p nodes (Observation 38.2) and there are p disjoint sets $S_1^*, S_2^*, \ldots, S_p^*$, there must be at least one set, say S_r^* (for some r, $1 \leq r \leq p$), such that $P_k \cap S_r^* = \emptyset$. Therefore, by the definition of S_r^*, we must have for each $u \in P_k$, $w(v_r^*, u) \geq \ell^*/2$. Since v_r^* is available for selection by GMM, and GMM selects a node $v \in V - P_k$ for which $\min_{v' \in P_k} w(v, v')$ is maximum among the nodes in $V - P_k$, it follows that Condition (38.3) holds after the $(k+1)$th addition to P. This completes the proof of Theorem 38.1. □

The next theorem provides a lower bound on the obtainable performance guarantee for MMFD:TI.

Theorem 38.2

If **P** $\neq$ **NP**, *no polynomial-time approximation algorithm can provide a performance guarantee of* $(2 - \epsilon)$ *for any* $\epsilon > 0$ *for MMFD:TI.*

Proof

We use a reduction from the MIS problem. Given an instance of the MIS problem consisting of graph $G(V, E)$ and an integer J, construct an instance of MMFD:TI as follows. The nodes of the MMFD:TI instance are in one-to-one correspondence with the nodes in V. For each edge $\{v_i, v_j\}$ in E, the weight $w(v_i, v_j)$ is chosen as 1. All the other weights are chosen as 2. Obviously, the resulting distances satisfy the triangle inequality. The number p of facilities to be chosen is set to J. If G contains an independent set V' of size at least J, then placing the facilities on the nodes of in V' provides a solution value of 2 for the MMFD:TI instance; otherwise, every placement has a solution value of 1. Therefore, any polynomial-time approximation algorithm $\mathcal{A}$ with a performance guarantee of $2 - \epsilon$ for any $\epsilon > 0$, for MMFD:TI will output a solution value of 2 if and only if G contains an independent set of size at least J. Thus, we obtain a polynomial-time algorithm for the MIS problem, contradicting the assumption that **P** $\neq$ **NP**. □

Theorems 38.1 and 38.2 indicate that the simple greedy algorithm of Figure 38.1 provides the best performance guarantee for the MMFD:TI problem that one can hope to obtain in polynomial time. A variant of this algorithm which can be used to approximate a more general form of MMFD, where facilities may be placed on nodes or edges, is analyzed in Ref. [10].

38.3.2.2 Results for Geometric Versions

Geometric versions of the MMFD problem have also been considered in the literature. We will first mention the results under the discrete placement model. The one-dimensional version of the problem (where all the points are on a line) can be solved in $O(pn + n \log n)$ time [16]. When the points are in two (or higher)-dimensional Euclidean space, the problem is **NP**-hard [16,29]. Since the triangle inequality holds for all geometric instances, the greedy algorithm of Figure 38.1 provides a performance guarantee of 2 for geometric instances of the MMFD problem under the discrete placement model.

Baur and Fekete [18] considered the MMFD problem and several of its variants under the continuous placement model, where the region is specified by a rectilinear polygon and the distance metric is L_1 or L_∞.

If the specified region is not required to be connected, they show that the MMFD problem cannot be approximated to within the factor $2 - \epsilon$ for any $\epsilon > 0$, unless $\mathbf{P} = \mathbf{NP}$. For a more general version of the problem, where the goal is to maximize the minimum distance between each pair of chosen points as well as the closest distance between a chosen point and the boundary of the region, they present an approximation algorithm with a performance guarantee of 3/2. They also show that the problem cannot be approximated to within the factor 14/13, unless $\mathbf{P} = \mathbf{NP}$. Additional approximation results for dispersional packing problems (e.g., finding the largest value of L such that a specified number of $L \times L$ squares can be packed in a given region) are also presented in Ref. [18].

38.3.3 Approximation Algorithms for Max-Average Dispersion

38.3.3.1 Results for Metric Instances

Recall that in the Max-Average dispersion (MAFD) problem, the goal is to maximize the average distance between a pair of chosen facilities. We use MAFD to denote this problem and MAFD:TI to denote its restriction to metric instances. We discuss approximation results for MAFD:TI in this section. Subsequent sections consider nonmetric and geometric instances of MAFD.

Reference [12] presents a modified version of the greedy approach used in Figure 38.1 to obtain an approximation algorithm for MAFD:TI. The modification is to replace step 3(a) in that figure by the following: Find a node $v \in V - P$ such that the average distance from v to the nodes in P is maximum among all the nodes in $V - P$. An inductive argument is used in Ref. [12] to show that this modification yields an approximation algorithm with a performance guarantee of 4.

Hassin et al. [30] present two approximation algorithms for MAFD:TI. Each of these algorithms provides a performance guarantee of 2. We will discuss one of their approximation algorithms below as it is also used in Ref. [14] for approximating another dispersion problem.

The details of the heuristic from Ref. [30], which we call Heuristic-HRT, are shown in Figure 38.2. The key step in the heuristic is the computation of a maximum-weight $\lfloor p/2 \rfloor$-matching in G (Step 2).

Input: A complete graph $G(V, E)$ with a nonnegative edge-weight $w(v_i, v_j)$ for each edge $\{v_i, v_j\} \in E$ and an integer $p \leq |V|$. (The edge weights satisfy the triangle inequality.)

Output: A set $P \subseteq V$ such that $|P| = p$. (The goal is to make the average edge weight in $G(P)$ close to the optimum value.)

Algorithm:

```
1. Let $P = \emptyset$.
2. Compute a $\lfloor p/2 \rfloor$-matching $M^*$ of maximum weight in $G$.
3. Add both end points of the edges in $M^*$ to $P$.
4. If $p$ is odd, add an arbitrary node from $V - P$ to $P$.
5. Output $P$.
```

FIGURE 38.2 Details of heuristic-HRT.

Using the fact that the problem of finding a maximum-weighted matching in any graph can be solved in polynomial time [31], it is shown in Ref. [30] that the problem of computing a maximum-weight $\lfloor p/2 \rfloor$-matching can also be solved efficiently. Thus, Heuristic-HRT runs in polynomial time. A careful implementation of the heuristic which leads to a running time of $O(n^2(p+\log n))$ is presented in Ref. [30]. We now prove the performance guarantee provided by the heuristic, closely following the presentation in Ref. [30].

The following notation is used in the proof. As before, for any nonempty node subset V_1, $G(V_1)$ denotes the complete subgraph of G induced on V_1. Further, $W(V_1)$ and $\overline{W}(V_1)$ denote, respectively, the total weight and the average weight of the edges in the complete subgraph $G(V_1)$. Similarly, for any nonempty subset E_1 of edges, $W(E_1)$ and $\overline{W}(E_1) = W(E_1)/|E_1|$ denote, respectively, the total and average weight of the edges in E_1. The following lemma establishes a property of maximum-weight $\lfloor p/2 \rfloor$-matchings.

Lemma 38.1

Let V_1 be a subset with at least $p \geq 2$ nodes and let M_1^* denote a maximum-weight $\lfloor p/2 \rfloor$-matching in $G(V_1)$. Then, $\overline{W}(V_1) \leq \overline{W}(M_1^*)$.

Proof

Let q denote the number of $\lfloor p/2 \rfloor$-matchings of $G(V_1)$, and let $M_1, M_2, \ldots, M_q$ denote the matchings themselves. Let $A_1 = \sum_{i=1}^{q} W(M_i)/q$ denote the average of the q values $W(M_1), W(M_2), \ldots, W(M_q)$. We will show that $A_1 = \lfloor p/2 \rfloor \overline{W}(V_1)$. Since $W(M_1^*) \geq A_1$, the lemma would then follow.

Consider the summation $\sum_{i=1}^{q} W(M_i)/q$. The number of edge weights included in this summation is $q\lfloor p/2 \rfloor$, since $|M_i| = \lfloor p/2 \rfloor$, for $1 \leq i \leq q$. By symmetry, the weight of each edge of $G(V_1)$ occurs the same number of times, say t, in the summation. Therefore, the number of occurrences of edge weights in the summation is also equal to $t\,|V_1|\,(|V_1| - 1)/2$. Hence,

$$q \lfloor p/2 \rfloor = t\,|V_1|\,(|V_1| - 1)/2 \tag{38.4}$$

Now,

$$A_1 = \sum_{i=1}^{q} W(M_i)/q = t\,W(V_1)/q = \frac{t\,\overline{W}(V_1)\,|V_1|\,(|V_1| - 1)}{2q} \tag{38.5}$$

Now, using Eq. (38.4), we get $A_1 = \lfloor p/2 \rfloor\, \overline{W}(V_1)$ as desired. □

The next lemma, which relies on the triangle inequality, shows a relationship between the average edge weight of a complete subgraph and the average weight of any $\lfloor p/2 \rfloor$-matching in the subgraph.

Lemma 38.2

Let V_1 be a subset containing $p \geq 2$ nodes and let M be any $\lfloor p/2 \rfloor$-matching in $G(V_1)$. Then, $\overline{W}(V_1) > \overline{W}(M)/2$.

Proof

Let $M = \{\{a_i, b_i\} : 1 \leq i \leq \lfloor p/2 \rfloor\}$ and let V_M denote the set of nodes which are endpoints of the edges in M. Consider each edge $\{a_i, b_i\}$ in M and let E_i denote the set of edges in $G(V_1)$ incident on a_i or b_i, except for the edge $\{a_i, b_i\}$ itself. By the triangle inequality, for any node $v \in V_M - \{a_i, b_i\}$, we have $w(v, a_i) + w(v, b_i) \geq w(a_i, b_i)$. When this inequality is summed up over all the nodes in $V_M - \{a_i, b_i\}$, we get

$$W(E_i) \geq (p - 2)\, w(a_i, b_i) \tag{38.6}$$

Now, we consider two cases.

Case 1. Let p be even. So, $\lfloor p/2 \rfloor = p/2$.

Consider the summation of Inequality (38.6) over all the edge sets E_i, $1 \leq i \leq p/2$. On the left-hand side of that summation, each edge of $G(V_1)$, except for those in M, appears twice. Therefore,

$2[W(V_1) - W(M)] \geq (p-2)\, W(M)$. Expressing this inequality in terms of $\overline{W}(V_1)$ and $\overline{W}(M)$, we get $\overline{W}(V_1) \geq p\, \overline{W}(M)/[2(p-1)]$; that is, $\overline{W}(V_1) > \overline{W}(M)/2$. This completes the proof for Case 1.

Case 2. Let p be odd. So, $\lfloor p/2 \rfloor = (p-1)/2$.

Let x be the node in $V_1 - V_M$, and let E_x denote the set of edges incident on x in $G(V_1)$. By the triangle inequality, we have

$$W(E_x) \geq W(M) \tag{38.7}$$

Again, consider the summation of Inequality (38.6) over the edge sets E_i, $1 \leq i \leq \lfloor p/2 \rfloor$. On the left-hand side of that summation, each edge of $G(V_1)$ appears twice, except that the edges in M do not appear at all and the edges in E_x appear only once. Therefore, $2[W(V_1) - W(M)] - W(E_x) \geq (p-2)\, W(M)$. Using Inequality (38.7), we get $2[W(V_1) - W(M)] \geq (p-1)\, W(M)$. Again expressing this inequality in terms of $\overline{W}(V_1)$ and $\overline{W}(M)$, and using the fact that $\lfloor p/2 \rfloor = (p-1)/2$, we get $\overline{W}(V_1) \geq (p+1)\, \overline{W}(M)/(2p)$. Consequently, $\overline{W}(V_1) > \overline{W}(M)/2$. This completes the proof for Case 2 and also that of the lemma. □

We are now ready to establish the performance guarantee of Heuristic-HRT.

Theorem 38.3

Let I be an instance of MAFD:TI. Let OPT(I) and HRT(I) denote the value of an optimal solution and that of the solution produced by Heuristic-HRT respectively. Then, OPT(I)/HRT(I) < 2.

Proof

Let P^* and P denote, respectively, the set of nodes in an optimal solution and that in the solution produced by Heuristic-HRT for the instance I. By definition, $OPT(I) = \overline{W}(P^*)$ and $HRT(I) = \overline{W}(P)$. Let M^* and M denote respectively a maximum-weight $\lfloor p/2 \rfloor$-matching in P^* and P. By Lemma 38.1, we have

$$OPT(I) \leq \overline{W}(M^*) \tag{38.8}$$

Further, by Lemma 38.2, we have

$$HRT(I) > \overline{W}(M)/2 \tag{38.9}$$

Since Heuristic-HRT computes a maximum-weight $\lfloor p/2 \rfloor$-matching for the graph G, we have $\overline{W}(M) \geq \overline{W}(M^*)$. This fact in conjunction with Inequalities (38.8) and (38.9) yields the theorem. □

A lower bound example is presented in Ref. [30] to show that the performance guarantee of Heuristic-HRT can be made arbitrarily close to 2.

38.3.3.2 Results for Nonmetric Instances

Here, we briefly mention the known results for nonmetric instances of the MAFD problem. When the weight of every edge in the complete graph G (which is part of the MAFD problem instance) is 0 or 1, the input can be thought of as a graph $G'(V, E')$, where E' contains only the edges of weight 1 in G. Then, the MAFD problem corresponds to the following problem: select a subset V' of p nodes such that the number of edges in the subgraph of G' induced on V' is a maximum over all induced subgraphs on p nodes. This problem is called the **Dense *p*-Subgraph** (D*p*S) problem in the literature. Feige et al. [32] present an approximation algorithm with a performance guarantee of $O(n^\delta)$, where $\delta < 1/3$, for the problem. They also show that the algorithm can be modified to obtain a performance guarantee of $O(n^\delta \log n)$ for a more general version of the problem, where the edges in G' have nonnegative edge weights and the goal is to choose a subset of p nodes so that the weight of all the edges in the chosen induced subgraph is maximized. When $p = \Omega(n)$, Asahiro et al. [33] show that there is an approximation algorithm with a constant performance guarantee for the D*p*S problem. When the graph G' is dense (i.e., the number of edges in $G' = \Omega(n^2)$) and $p = \Omega(n)$, there is a PTAS for the D*p*S problem [34]. When the edge weights in the input graph are from the set $\{1, 2, \ldots, K\}$, for some fixed positive integer K and the number of edges in an optimal solution is $\Omega(n^2)$, Czygrinow [35] presents a faster PTAS for the weighted version of the

DpS problem. Feige [36] shows the existence of a constant $\rho > 1$ such that if there is a polynomial-time ρ-approximation algorithm for the DpS problem, then a natural conjecture regarding the average case hardness of the Boolean Satisfiability problem would be violated. No stronger nonapproximability results are currently known for the DpS problem.

38.3.3.3 Geometric Versions

Geometric versions of the MAFD problem seem to have been considered only under the discrete placement model. The approximation algorithm for the MAFD:TI problem presented in Section 38.3.3.1 provides a performance guarantee of 2 for geometric versions. Better performance guarantee results have been obtained by using ideas from computational geometry. In Refs. [12,28] it is shown that the one-dimensional version of the MAFD problem can be solved efficiently. Using this result, an approximation algorithm for the two-dimensional MAFD problem under Euclidean distances was presented in Ref. [12]. The basic idea of the approximation algorithm is to create a polynomial number of instances of the one-dimensional MAFD problem by projecting the given set of points onto an appropriate set of lines, solve each one-dimensional problem optimally and choose the best of these solutions. It is shown that this scheme provides an asymptotic performance guarantee[4] of $\pi/2 \approx 1.571$ for the two-dimensional MAFD problem.

Fekete and Meijer [17] consider the MAFD problem under rectilinear distances for points in d-dimensional space, for any fixed $d \geq 2$. When p, the number of points to be selected, is fixed, they show that the MAFD problem can be solved in $O(n)$ time. When p is part of the problem instance, they present a PTAS for the problem. By appropriately modifying this PTAS, they show that an approximation algorithm with an asymptotic performance guarantee of $\sqrt{2}$ can be obtained for the case of Euclidean distances in two-dimensional space, thus improving the asymptotic bound of $\pi/2$ in Ref. [12].

38.3.4 Results for Other Dispersion Problems

Approximation results for a number of other dispersion objectives have been reported in the literature [13,14]. In particular, results for objectives Max-MST, Max-ST, and Max-TSP are presented in Ref. [13] while results for Max-Matching and Max-Star are presented in Ref. [14]. Below, we summarize these results.

Halldórsson et al. [13] consider dispersion problems under three objectives, namely minimum weight spanning tree (Max-MST), minimum TSP tour length (Max-TSP) and minimum ST (Max-ST). For metric instances of these three problems, they show that the greedy algorithm discussed in Section 38.3.2.1 provides performance guarantees of 4, 3, and 3, respectively. Under the assumption that $\mathbf{P} \neq \mathbf{NP}$, they also show lower bounds of 2, 2, and 4/3, respectively on the performance guarantees obtainable in polynomial time for the three problems. Since constructing a near-minimum ST is done using shortest path distances between nodes, the greedy approach provides a performance guarantee of 3 for the Max-ST dispersion problem even for nonmetric instances. However, nonmetric instances of Max-MST and Max-TSP are shown to be at least as hard to approximate as the MIS problem. In Ref. [13], geometric versions of dispersion problems under the discrete placement model are also considered. For two-dimensional versions of Max-MST and Max-ST, where the distance between two points is the Euclidean distance, they present approximation algorithms with performance guarantees of 2.25 and 2.16, respectively.

In Ref. [14], an approximation algorithm with a performance guarantee of $O(\log p)$ for the metric instances of the Max-Matching problem was provided. They obtain the algorithm by suitably modifying the greedy approach (used in Section 38.3.2.1). For metric instances of the Max-Star problem, they show that the HRT-heuristic [30] (discussed in Section 38.3.3.1) provides a performance guarantee of 2. They also discuss other applications of the HRT-heuristic.

Further, Chandra and Halldórsson [14] carry out a more detailed study of the greedy approach for approximating dispersion problems. They show that for many dispersion objectives, the ratio of the

[4] An algorithm provides an **asymptotic** performance guarantee of ρ, if for any fixed $\epsilon > 0$, the performance guarantee of the algorithm is at most $\rho + \epsilon$.

optimal solution value to the solution value produced by the greedy approach may be arbitrarily large. They also show that the greedy approach is useful for a more general version of the Max-MST, Max-TSP, and Max-ST dispersion problems. In this general version, an integer $k \leq p$ is also specified as part of the input. After an algorithm chooses a subset of p nodes, an adversary partitions the chosen subset into k subsets, and the value of the objective is the sum of the weights of the corresponding minimum structure (MST, minimum TSP tour or minimum ST) on the k subgraphs. For the generalized version of the Max-MST problem, it is shown that the greedy approach provides a performance guarantee of $4-2/(p-k+1)$. For the generalized versions of Max-TSP and Max-ST problems, the greedy approach is shown to provide a performance guarantee of $\min\{5,\ 2p/(p-k)\}$.

38.4 Approximation Algorithms for Capacitated Versions

38.4.1 Motivation and Problem Definition

The model of dispersion considered in previous sections does not explicitly consider the storage capacity of a node. In general, different nodes may have different capacities. This practical aspect adds a new dimension and leads to **capacitated dispersion problems** [27]. These are typically constrained optimization problems in which the constraints or the optimization objectives involve capacities of nodes. One example of such a problem is the following: Choose a subset of nodes so that the minimum distance between any pair of chosen nodes is at least a specified threshold and the total storage capacity of the chosen nodes is a maximum over all subsets that satisfy the distance constraint. The focus of this section is on approximation algorithms for such capacitated dispersion problems.

The following notation is used throughout this section. For a subset of nodes V', $CAP(V')$ denotes the sum of the storage capacities of the nodes in V'. For any subset of nodes V' with $|V'| \geq 2$, $DIST(V')$ denotes the *minimum* distance between a pair of nodes in V'; this is the Max-Min dispersion measure for the set V'.

Formulations of some capacitated dispersion problems are given below. In naming these problems, we use the following convention. Each problem has a maximization objective and one or more constrained measures. The maximization objective is indicated by the prefix "Max," and each constrained measure is preceded by a slash ("/"). For example, in the Max-Cap/Dist problem, the objective is to maximize the total capacity and the constraint is on the internode distance. A precise formulation of this problem is as follows.

Maximizing Capacity under a Distance Constraint (Max-Cap/Dist)

Instance

A complete graph $G(V, E)$ with an edge weight $w(v_i, v_j)$ for each edge $\{v_i, v_j\}$ in E, a storage capacity c_i (rational number) for each node $v_i \in V$ and a positive rational number α.

Requirement

Select a subset $V' \subseteq V$ so that CAP(V') is a maximum over all subsets V' that satisfy the constraint DIST$(V') \geq \alpha$.

The dual[5] version of Max-Cap/Dist, where a required capacity value B is to given and the goal is to select a subset V' of nodes so that DIST(V') is maximum among all subsets satisfying the constraint CAP$(V') \geq B$, is denoted by Max-Dist/Cap. We will use Max-Cap/Dist:TI (Max-Dist/Cap:TI) to denote the restriction of Max-Cap/Dist (Max-Dist/Cap) to metric instances.

Geometric versions of the capacitated dispersion problems seem to have been considered only for the discrete placement model [27]. In our notation, the geometric version of a problem will be specified with the appropriate dimension as the prefix. For example, the one- and two-dimensional versions of the

[5] We use the term "dual" in a narrow sense to mean that the maximization objective and constrained measure are interchanged.

Max-Cap/Dist problem will be denoted by 1D:Max-Cap/Dist and 2D:Max-Cap/Dist, respectively. In all problems involving a capacity constraint, we assume without loss of generality that no single node has a large enough capacity to satisfy the constraint.

38.4.2 Approximation Results for Capacitated Dispersion

38.4.2.1 A Nonapproximability Result for Maximizing Capacity

The Max-Cap/Dist:TI problem is at least as hard to approximate as the MIS problem. This can be seen as follows. Given an instance of the MIS problem represented by an undirected graph $G(V, E)$, construct a Max-Cap/Dist:TI instance as follows. Treat V as the set of nodes, each with a capacity of one unit. For any pair of nodes u and v, let $w(u, v) = 1$ if $\{u, v\}$ is an edge in G and let $w(u, v) = 2$ otherwise. Obviously, the resulting distances satisfy the triangle inequality. Let the minimum distance constraint be set to 2. It can be verified that any feasible solution of capacity γ to the constructed Max-Cap/Dist:TI instance corresponds to an independent set of size γ in G and vice versa. As a consequence, nonapproximability results for the MIS problem (mentioned in Section 38.2.2) carry over to the Max-Cap/Dist:TI problem. The following proposition provides a formal statement of this result.

Proposition 38.1

Unless **NP** $=$ **ZPP**, *there is no polynomial-time approximation algorithm with a performance guarantee of* $O(n^{1-\epsilon})$ *for any* $\epsilon > 0$ *for the Max-Cap/Dist:TI problem.*

38.4.2.2 Results for Maximizing Internode Distance

The approximation problem for nonmetric instances of Max-Dist/Cap is at least as hard as that for the nonmetric instances of Max-Min dispersion (MMFD). To see this, consider an instance of the MMFD problem with n nodes and let p denote the number of facilities to be placed. Construct an instance of the Max-Dist/Cap problem as follows. The two instances have the set of nodes and internode distances. The capacity of each node is set to 1 and the capacity requirement is set to p. Now, it is straightforward to verify that any solution to the MMFD instance with a Max-Min objective value of α is a feasible solution with the same objective value for the Max-Dist/Cap instance. As a consequence, the known nonapproximability result for nonmetric instances of MMFD (mentioned in Section 38.3.2.1) is applicable to the Max-Cap/Dist problem. This result is stated formally below.

Proposition 38.2

Unless **P** $=$ **NP**, *for any* $\rho \geq 1$, *there is no polynomial-time* ρ*-approximation algorithm for nonmetric instances of the Max-Dist/Cap problem.*

We now consider metric instances of the Max-Dist/Cap problem, denoted by Max-Dist/Cap:TI. A further restricted version of this problem, where all nodes have a capacity of 1, is the MMFD:TI problem considered in Section 38.3.2. Thus, there is a 2-approximation algorithm for this restricted version of Max-Dist/Cap:TI problem, and this approximation cannot be improved unless **P** $=$ **NP**. We now show that the more general version of the Max-Dist/Cap:TI problem, where nodes have arbitrary capacities, can also be approximated to within a factor of 2.

Recall that the specification of the Max-Dist/Cap:TI problem includes a lower bound B on the required capacity. To ensure feasibility, we assume that the sum of the capacities of all the nodes in G is at least B. As mentioned earlier, we also assume that no node has a capacity of B or more. Our heuristic for Max-Dist/Cap:TI is shown in Figure 38.3. It is based on a binary search over the internode distances. For each query distance α, the heuristic invokes procedure Greedy (also shown in Figure 38.3) to try to find a subset of nodes V' for which $DIST(V') \geq \alpha$ and $CAP(V') \geq B$. It will be shown (Lemma 38.3) that procedure Greedy returns "success" (and the corresponding placement V') for any internode distance which is at least half the optimal value.

To establish the performance guarantee provided by the heuristic, we have the following lemma.

Input: A complete graph $G(V, E)$ with edge weights satisfying the triangle inequality, a capacity c_i for each node $v_i \in V$ and a capacity requirement B.

Output: A subset V' of V such that $\text{CAP}(V') \geq B$. (The goal is to make $\text{DIST}(V')$ close to the optimum value.)

Algorithm:

```
1. Sort the nodes in nonincreasing capacity order, and create a list, denoted by SiteList.
2. Sort the internode distances in nonincreasing order and eliminate duplicate distances. Let the resulting
   sorted list of distances be stored in the array $D[1 .. t]$ such that $D[1] > D[2] > \cdots > D[t]$.
3. Carry out a binary search over the array $D$ to find the index $i$ such that for $\alpha = D[i]$, the call
   Greedy($\alpha$, $B$) returns "success" and for $\alpha' = D[i+1]$, the call Greedy ($\alpha'$, $B$) returns "failure."
4. Output the internode distance $\alpha$ along with the corresponding placement $V'$ found in step 3.

procedure Greedy ($\alpha$, $B$)
  (a) Let $L$ = SiteList and $V' = \emptyset$.
  (b) while $L$ is not empty do
        (i) Add the first node $v$ from $L$ to $V'$.
        (ii) Remove from $L$ all nodes (including $v$) whose distance to $v$ is strictly less than $\alpha$.
  (c) if $CAP(V') \geq B$ then return "success" and the set $V'$ else return "failure."
```

FIGURE 38.3 Details of the heuristic for Max-Dist/Cap:TI.

Lemma 38.3

Let I denote any instance of Max-Dist/Cap:TI problem for which there is a feasible solution. Let V^* denote an optimal set of nodes for the instance I and let $OPT(I) = DIST(V^*)$. Let γ be the smallest internode distance $\geq OPT(I)/2$. For any internode distance $\alpha \leq \gamma$, the call Greedy(α, B) returns "success."

Proof

Let $V^* = \{v_{i_1}, v_{i_2}, \ldots, v_{i_r}\}$ denote the nodes in the optimal solution. Without loss of generality, we assume that $c_{i_1} \geq c_{i_2} \geq \cdots \geq c_{i_r}$. Note that for any two nodes v_{i_a} and v_{i_b} in V^*, $w(v_{i_a}, v_{i_b}) \geq OPT(I)$.

Consider the call Greedy(α, B). Let v_{j_1} be the first node added to V' in this call. Since the procedure considers nodes in nonincreasing order of capacities, we have $c_{j_1} \geq c_{i_1}$. We now have the following claim.

Claim
$w(v_{j_1}, v_{i_1}) \geq \alpha$ or $w(v_{j_1}, v_{i_2}) \geq \alpha$.

Proof
Suppose the claim is false; that is, $w(v_{j_1}, v_{i_1}) < \alpha$ and $w(v_{j_1}, v_{i_2}) < \alpha$. Since $\alpha \leq \gamma$ and γ is the smallest internode distance $\geq OPT(I)/2$, it follows that $w(v_{j_1}, v_{i_1}) < OPT(I)/2$ and $w(v_{j_1}, v_{i_2}) < OPT(I)/2$. Consequently, $w(v_{j_1}, v_{i_1}) + w(v_{j_1}, v_{i_2}) < OPT(I)$. However, since v_{i_1} and v_{i_2} are both in the optimal solution V^*, $w(v_{i_1}, v_{i_2}) \geq OPT(I)$. This violates the triangle inequality, and the claim follows. □

During the call Greedy(α,B), after selecting v_{j_1}, only those nodes which are at a distance strictly less than α are removed from L. Thus, from the above claim, at most one of the nodes v_{i_1} and v_{i_2} is eliminated when v_{j_1} is chosen. Consequently, a node of capacity at least c_{i_2} is available for selection during the second iteration.

A straightforward extension of the above argument shows that for $1 \leq q \leq r$, where r is the number of nodes in the optimal solution V^* under consideration, a node of capacity at least c_{i_q} is available for selection during iteration q of the **while** loop of procedure Greedy. As a consequence, the call Greedy(α,B) will select a set V' of nodes with CAP(V') $\geq B$ and return "success." □

Theorem 38.4

Let I denote any instance of Max-Dist/Cap:TI problem for which there is a feasible solution. Let V^* denote an optimal set of nodes for the instance I and let $OPT(I) = DIST(V^*)$. Let $HEU(I)$ be the distance output by the heuristic shown in Figure 38.3. Then $OPT(I)/HEU(I) \leq 2$.

Proof
Because of the binary search used in the heuristic, the distance value α returned by the heuristic is such that Greedy(α, B) returns "success," but if α' is the next largest distance after α, then Greedy(α', B) returns "failure." By Lemma 38.3, $\alpha \geq \gamma$, the smallest internode distance $\geq OPT(I)/2$. □

A lower bound example is presented in Ref. [27] to show that the bound of 2 established in the above theorem is tight. It can be seen that the running time of the heuristic in Figure 38.3 is $O(n^2 \log n)$.

38.4.3 Results for Geometric Versions

All the results mentioned in this section are for the discrete placement model. Problems 1D:Max-Cap/Dist and 1D:Max-Dist/Cap can be solved in polynomial time [27] and hence are not discussed further. Using a straightforward reduction from the MIS problem for unit disk graphs, both 2D:Max-Cap/Dist and 2D:Max-Dist/Cap can be shown to be **NP**-complete, even when each node has a capacity of 1.

As mentioned in Section 38.2.2, a PTAS is known for the weighted MIS problem for unit disk graphs when the geometric representation is available as part of the input [26,27]. Using this result, a PTAS for the 2D:Max-Cap/Dist problem can be obtained. This is done by reducing the 2D:Max-Cap/Dist problem to the weighted MIS problem for unit disk graphs in the following manner. Treat each given point in the 2D:Max-Cap/Dist problem as the center of a unit disk with radius = $\alpha/2$, where α is the distance constraint. The weight of each disk is the storage capacity of the corresponding point. Now, it can be seen that any independent set of weight γ for the constructed unit disk graph is a feasible solution of capacity γ for the 2D:Max-Cap/Dist problem. Therefore, there is a PTAS for the 2D:Max-Cap/Dist problem.

A heuristic with a performance guarantee of 2 for the Max-Dist/Cap:TI was presented in Section 38.4.2. Since the internode distances in any instance of 2D:Max-Dist/Cap satisfy the triangle inequality, it follows that there is a 2-approximation algorithm for the 2D:Max-Dist/Cap problem. The following result summarizes the above discussion.

Proposition 38.3

(a) *There is a PTAS for the 2D:Max-Cap/Dist problem.*
(b) *There is a 2-approximation algorithm for the 2D:Max-Dist/Cap problem.* ■

38.4.4 Capacitated Dispersion with Storage Costs

In Ref. [27] capacitated dispersion problems with storage costs were also considered. Their model assumes that the storage cost at a node is a linear function of the amount of material stored at the node. Such dispersion problems, which involve three measures, namely capacity, distance, and cost, are formulated by choosing one of these measures as the optimization objective and specifying constraints on the other two measures. The formulation leads to minimization problems when the optimization objective is cost and to maximization problems when the optimization objective is either capacity or distance. Likewise, the constraint on cost specifies an upper bound (budget) while constraints on the other two measures specify lower bounds (minimum required value). A solution to such a problem consists of a placement along with the amount of material stored at each node in the placement. In specifying these problems, we use the notation introduced in Section 38.4.1, where the optimization objective (preceded by "Min" or "Max") is given first and each constrained measure is preceded by a "/." For example, in the Min-Cost/Cap/Dist problem, the objective is to minimize the storage cost while satisfying constraints on capacity and minimum internode distance. In the remainder of this section, we will summarize the known results for such problems.

In general, many of the problems involving storage costs are difficult to approximate. For example, strong nonapproximability results are known for metric instances of Min-Cost/Cap/Dist, Max-Dist/Cap/Cost, and Max-Cap/Cost/Dist problems. Further, the 2D:Min-Cost/Cap/Dist problem has no ρ-approximation for any $\rho \geq 1$, unless $\mathbf{P} = \mathbf{NP}$. A similar nonapproximability result is also known for the 1D:Max-Dist/Cap/Dist problem. Positive results are known only for geometric versions of some of the problems. For example, there is a PTAS for the 1D:Min-Cost/Cap/Dist problem. This PTAS is obtained by starting with a pseudopolynomial algorithm[6] that solves the problem optimally, and converting the algorithm into a PTAS using standard scaling and rounding techniques [15]. Also, a PTAS is known for the 2D:Max-Cap/Dist/Cost problem. This PTAS is obtained by combining the known PTAS for the weighted MIS problem for unit disk graphs [26] with another PTAS for the problem of finding a solution of maximum capacity under a cost constraint. Details regarding the above results can be found in Ref. [27].

38.5 Future Directions

In the previous sections, we discussed approximation algorithms for several versions of dispersion problems. We conclude this chapter by presenting some directions for future research.

For the metric instances of MAFD, the best known approximation algorithm [30] provides a performance guarantee of 2. Whether this performance guarantee can be improved remains an open problem. For the nonmetric instances of MAFD, there is a significant gap between the upper bound of $O(n^{\delta} \log n)$ [32] and the lower bound of $O(1)$ [36] on the performance guarantee. Narrowing this gap is an interesting research problem. For metric instances of the Max-Matching dispersion problem, an approximation algorithm with a performance guarantee of $O(\log p)$ is known [14], but no lower bounds on the achievable performance guarantee are known. Many of the known results for the geometric versions of dispersion problem are for the discrete placement model. Under the continuous placement model, results are available for the Max-Min objective [18]. Investigating other dispersion problems under the continuous placement model is a useful research direction. The topic of capacitated dispersion also provides several open questions. For example, all the known results for capacitated dispersion [27] use the Max-Min measure for the internode distance. It is of interest to consider capacitated dispersion problems for other distance measures (e.g., Max-Average distance). Dispersion problems involving multiple types of facilities are of practical importance. Exploring approximation algorithms for such problems would be a fruitful research endeavor.

Another research direction is offered by the **online** model for facility placement. In this model, introduced in Ref. [37], the distance between each pair of nodes is given as part of the input, but p, the number

[6] A **pseudopolynomial** algorithm has a running time that is a polynomial function of the size of the input and the maximum value that appears in the input [15].

of facilities to be placed is not known a priori. Instead, requests for placing facilities arrive one at a time. An algorithm must choose a new node for each request. Previously placed facilities cannot be moved or eliminated. Under this online model, constant factor approximations were presented in Ref. [37] for location problems involving desirable facilities. Using the same online model, some results for dispersion problems are presented in Ref. [38]. In particular, it is shown that for dispersion objectives satisfying certain properties, approximation algorithms for the offline model (where the value of p is known a priori) can be used to obtain approximation algorithms under the online model, with only an $O(1)$ loss in the performance guarantee. This result also holds for nonmetric instances. Results on **robust subgraphs** developed in Ref. [39] are also germane to the context of online facility placement problems. In general, developing approximation algorithms for facility dispersion problems under the online model remains an interesting research direction.

References

[1] Mirchandani, P. B. and Francis, R. L., *Discrete Location Theory*, Wiley, New York, NY, 1990.

[2] Erkut, E. and Neuman, S., Analytical models for locating undesirable facilities, *Eur. J. Oper. Res.*, 40(3), 275, 1989.

[3] Curtin, K. M. and Church, R. L., A Family of Models for Multiple-Type Discrete Dispersion, Political Economy Working paper 34/03, School of Social Sciences, University of Texas at Dallas, August 2003.

[4] Kuby, M. J., Programming models for facility dispersion: the p-dispersion and maxisum dispersion problems, *Geog. Anal.*, 19(4), 315, 1987.

[5] Steuer, R. E., *Multiple Criteria Optimization: Theory and Application*, Wiley, New York, 1986.

[6] Church, R. L. and Garfinkel, R. S., Locating an obnoxious facility on a network, *Transp. Sci.*, 12(2), 107, 1978.

[7] Chandrasekharan, R. and Daughety, A., Location on tree networks: p-centre and n-dispersion problems, *Math. Oper. Res.*, 6(1), 50, 1981.

[8] Moon, I. D. and Chaudhry, S. S., An analysis of network location problems with distance constraints, *Manage. Sci.*, 30(3), 290, 1984.

[9] Melachrinoudis, E. and Cullinane, T. P., Locating an undesirable facility with a minimax criterion, *Eur. J. Ope. Res.*, 24(2), 239, 1986.

[10] Tamir, A., Obnoxious facility location on graphs, *SIAM J. Disc. Math.*, 4(4), 550, 1991.

[11] White, D. J., The maximal dispersion problem and the 'first point outside the neighborhood' heuristic, *Comput. Oper. Res.*, 18(1), 43, 1991.

[12] Ravi, S. S., Rosenkrantz, D. J., and Tayi, G. K., Heuristic and special case algorithms for dispersion problems, *Oper. Res.*, 42(2), 299, 1994.

[13] Halldórsson, M. M., Iwano, K., Katoh, N., and Tokuyama, T., Finding subsets maximizing minimum structures, *SIAM J. Disc. Math.*, 12(3), 342, 1999.

[14] Chandra, B. and Halldórsson, M. M., Facility dispersion and remote subgraphs, *J. Algorithms*, 38(2), 438, 2001.

[15] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, CA, 1979.

[16] Wong, D. W. and Kuo, Y. S., A study of two geometric location problems, *Inf. Proc. Lett.*, 28(6), 281, 1988.

[17] Fekete, S. P. and Meijer, H., Maximum dispersion and geometric maximum weight cliques, *Algorithmica*, 38(3), 501, 2003.

[18] Baur, C. and Fekete, S. P., Approximation of geometric dispersion problems, *Algorithmica*, 30(3), 451, 2001.

[19] West, D. B., *Introduction to Graph Theory*, 2nd ed., Prentice-Hall, Inc., Englewood Cliffs, NJ, 2001.

[20] Cormen, T., Leiserson, C. E., Rivest, R., and Stein, C., *Introduction to Algorithms*, 2nd ed., MIT Press/McGraw-Hill, Cambridge, MA, 2001.

[21] Håstad, J., Clique is hard to approximate within $n^{1-\epsilon}$, *Acta Mathematica* 182, 105, 1999.

[22] Papadimitriou, C. H., *Computational Complexity*, Addison-Wesley Publishing Co., Reading, MA, 1994.

[23] Clark, B. N., Colbourn, C. J., and Johnson, D. S., Unit disk graphs, *Disc. Math.*, 86(1–3), 165, 1990.

[24] Breu, H. and Kirkpatrick, D. G., Unit disk graph recognition is NP-hard, *Comp. Geom.*, 9(1–2), 3, 1988.

[25] Marathe, M. V., Breu, H., Hunt, H. B., III, Ravi, S. S., and Rosenkrantz, D. J., Simple heuristics for unit disk graphs, *Networks*, 25(2), 59, 1995.

[26] Hunt, H. B., III, Marathe, M. V., Radhakrishnan, V., Ravi, S. S., Rosenkrantz, D. J., and Stearns, R. E., NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs, *J. Algorithms*, 26(2), 238, 1998.

[27] Rosenkrantz, D. J., Tayi, G. K., and Ravi, S. S., Facility dispersion problems under capacity and cost constraints, *J. Comb. Opt.*, 4(1), 7, 2000.

[28] Tamir, A., Comments on the paper: 'Heuristic and Special Case Algorithms for Dispersion Problems' by Ravi, S. S., Rosenkrantz, D. J., and Tayi, G. K., *Oper. Res.*, 46(1), 157, 1998.

[29] Fowler, R. J., Patterson, M., and Tanimoto, S. L., Optimal packing and covering in the plane are NP-complete, *Inf. Proc. Lett.*, 12(3), 133–137, 1981.

[30] Hassin, R., Rubinstein, S., and Tamir, A., Approximation algorithms for maximum dispersion, *Oper. Res. Lett.*, 21(3), 133, 1997.

[31] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, Inc., Upper Saddle River, NJ, 1993.

[32] Feige, U., Kortsarz, G., and Peleg, D., The dense *k*-subgraph problem, *Algorithmica*, 29(3), 410, 2001.

[33] Asahiro, Y., Iwama, K., Tamaki H., and Tokuyama, T., Greedily finding a dense subgraph, *J. Algorithms*, 34(2), 203, 2000.

[34] Arora, S., Karger, D., and Karpinski, M., polynomial time approximation schemes for dense instances of NP-hard problems, *JCSS*, 58(1), 193, 1999.

[35] Czygrinow, A., Maximum dispersion problem in dense graphs, *Oper. Res. Lett.*, 27(5), 223, 2000.

[36] Feige, U., Relations between average case complexity and approximation complexity, *Proc. STOC*, 2002, p. 534.

[37] Mettu, R. R. and Plaxton, C. G., The online median problem, *SIAM J. Comput.*, 32(3), 816, 2003.

[38] Rosenkrantz, D. J., Tayi, G. K., and Ravi, S. S., Obtaining Online Approximation for Facility Dispersion from Offline Algorithms, *Networks*, 47(4), 206, 2006.

[39] Hassin, R. and Segev, D., Robust subgraphs for trees and paths, *Proc. Scandinavian Workshop on Algorithm Theory*, Humlebaek, Denmark, 2004, p. 51.

39

Greedy Algorithms for Metric Facility Location Problems

Anthony Man-Cho So
Stanford University

Yinyu Ye
Stanford University

Jiawei Zhang
New York University

39.1 Introduction

Variants of the facility location problem (FLP) have been studied extensively in the operations research and management science literatures and have received considerable attention in the area of approximation algorithms. In the metric uncapacitated facility location problem (UFLP), which is the most basic facility location problem, we are given a finite set $\mathcal{F}$ of *facilities*, a finite set $\mathcal{D}$ of *clients*, a cost f_i for opening facility $i \in \mathcal{F}$, and a connection cost c_{ij} for connecting i to j for $i,\ j \in \mathcal{D} \cup \mathcal{F}$. The objective is to open a subset of the facilities in $\mathcal{F}$ and connect each client to an open facility so that the total cost is minimized. We assume that the connection costs form a metric, meaning that they are nonnegative, symmetric, and satisfy the triangle inequality, that is, for any $i,\ j,\ k \in \mathcal{D} \cup \mathcal{F}$, $c_{ij} \geq 0$, $c_{ij} = c_{ji}$, and $c_{ij} \leq c_{ik} + c_{kj}$.

More precisely, the problem can be formulated as an integer program (39.1)–(39.4):

$$\text{minimize} \quad \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{D}} c_{ij}x_{ij} + \sum_{i\in\mathcal{F}} f_i y_i \tag{39.1}$$

$$\text{subject to} \quad \sum_{i\in\mathcal{F}} x_{ij} = 1 \quad \text{for all} \quad j \in \mathcal{D} \tag{39.2}$$

$$x_{ij} \leq y_i \quad \text{for all} \quad i \in \mathcal{F},\ j \in \mathcal{D} \tag{39.3}$$

$$x_{ij},\ y_i \in \{0, 1\} \quad \text{for all} \quad i \in \mathcal{F},\ j \in \mathcal{D} \tag{39.4}$$

where the binary variable y_i indicates whether facility i is open and the binary variable x_{ij} indicates whether client j is connected to facility i. Constraints (39.2) ensure that each client is connected to a facility, whereas constraints (39.3) ensure that the clients are connected only to open facilities. This integer programming formulation has been attributed to Balinski [1].

We briefly review recent work on approximation algorithms for the metric UFLP. First of all, we note that a result of Guha and Khuller [2], combined with an observation of Sviridenko (cited as private communication in Ref. [3]), implies that no polynomial-time algorithm for the uncapacitated problem

TABLE 39.1 Approximation Algorithms for the Metric UFLP

Approximation factor	Reference	Technique/running time
$O(\ln \lvert\mathcal{D}\rvert)$	Hochbaum [4]	Greedy algorithm/$O(n^3)$
3.16	Shmoys et al. [5]	LP rounding
2.41	Guha and Khuller [2]	LP rounding
1.736	Chudak and Shmoys [6]	LP rounding
$5+\epsilon$	Korupolu et al. [8]	Local search/$O(n^6 \log(n/\epsilon))$
3	Jain and Vazirani [14]	Primal-dual method/$O(n^2 \log\ n)$
1.853	Charikar and Guha [9]	Primal-dual method/$O(n^3)$
1.728	Charikar and Guha [9]	LP rounding + primal-dual method
1.861	Mahdian et al. [18]	Greedy algorithm/$O(n^2 \log n)$
1.61	Jain et al. [19]	Greedy algorithm/$O(n^3)$
1.582	Sviridenko [7]	LP rounding
1.52	Mahdian et al. [20]	Greedy algorithm/$\tilde{O}(n)$

can have a performance guarantee better than 1.463 unless $P = NP$. Therefore, under the assumption that $P \neq NP$, we cannot expect a ρ-approximation algorithm with $\rho < 1.463$ for the metric UFLP and its generalization.

The first approximation algorithm for the uncapacitated problem was developed 20 years ago by Hochbaum [4] with performance guarantee $O(\ln \lvert\mathcal{D}\rvert)$. The first constant factor approximation algorithm was given by Shmoys et al. [5], who presented a 3.16-approximation algorithm based on rounding an (fractional) optimal solution of the linear programming (LP) relaxation of (39.1)–(39.4). Such an approach is called *linear programming rounding*. Subsequently, many approximation algorithms with better performance guarantees have been proposed. The approximation ratios and running times of these algorithms are summarized in Table 39.1, where $n = \lvert\mathcal{D}\rvert + \lvert\mathcal{F}\rvert$. These algorithms fall into three classes.

The first class contains approximation algorithms that are based on LP rounding, including Guha and Khuller [2], Chudak and Shmoys [6], and Sviridenko [7]. The algorithms in Refs. [2,5] based on solving LP relaxation of (39.1)–(39.4), while the algorithms in Refs. [6,7] also need to solve the dual of the LP relaxation.

The second class contains local search algorithms including Korupolu et al. [8], Charikar and Guha [9], and Arya et al. [10]. Starting with any feasible solution, the local search algorithms try to improve the current solution by opening new facility, closing an open facility, or swap two facilities (close one and open one), etc. These algorithms have been generalized to solve capacitated facility location problems where each facility can serve certain amount of demands [11–13].

The last class of algorithms is primal-dual in flavor. The first such algorithm was proposed by Jain and Vazirani [14]. The algorithms of Mettu and Plaxton [15] and Thorup [16] also fall into this class.

The greedy algorithms presented in this chapter can also be presented as dual-ascent algorithms. They are due to Jain et al. [17] (this is the journal version of two conference papers [18,19], and in many cases, we will refer to the conference papers) and Mahdian et al. [20]. We also present a generalization of the algorithm in Ref. [17] for a two-level facility location problem [21]. In the next section, we present the simplest greedy algorithm, and introduce the ideas of dual-fitting and factor-revealing LP, which are essential in analyzing the algorithm and other greedy algorithms for the metric UFLP. In Section 39.3, we present the more refined algorithms and the concept of bifactor approximation. Section 39.4 is devoted to the two-level facility location problem where we introduce the so-called quasi-greedy algorithm.

39.2 Dual-Fitting and Factor-Revealing LP

In greedy algorithms, at each step, one computes a greedy function value for each element of a candidate set and chooses the optimal candidate based on these values. For the metric UFLP, a natural choice for the candidate set is a *star* and the greedy function is related to the notion of *cost-effectiveness*. A star $S = (i, C)$

consists of one facility i and a set of clients C. The set of all stars is denoted by $\mathcal{S}$. For a star $S = (i, C)$, the cost of the star is defined by $c_S = f_i + \sum_{j \in C} c_{ij}$, and the cost-effectiveness of the star is defined by $c_S/|C|$.

Now we are ready to present our first algorithm, which was originally proposed by Mahdian et al. [18]. We call it the MMSV algorithm.

The MMSV Algorithm.

1. Let U be the set of unconnected clients. Initially $U = \mathcal{D}$.
2. While $U \neq \emptyset$:
   (a) Choose the most cost-effective star $S = (i, C)$ such that $C \subset U$, open facility i if it is not already open, and assign all clients in C to i.
   (b) Set $f_i := 0$, $U := U \setminus C$.

The algorithm is extremely simple and it is easy to implement. The performance guarantee of the algorithm is 1.861 and it has been reported that it performs very well on average [17].

The analysis of the algorithm relies on the dual of the LP relaxation of the UFLP, which can be represented as follows:

$$\text{maximize} \quad \sum_{j \in \mathcal{D}} \alpha_j \tag{39.5}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{D}} \beta_{ij} \leq f_i \quad \text{for all} \quad i \in \mathcal{F} \tag{39.6}$$

$$\alpha_j - \beta_{ij} \leq c_{ij} \quad \text{for all} \quad i \in \mathcal{F},\ j \in \mathcal{D} \tag{39.7}$$

$$\beta_{ij}, \alpha_j \geq 0 \quad \text{for all} \quad i \in \mathcal{F},\ j \in \mathcal{D} \tag{39.8}$$

It is equivalent to

$$\text{maximize} \quad \sum_{j \in \mathcal{D}} \alpha_j \tag{39.9}$$

$$\text{subject to} \quad \sum_{j \in S \cap \mathcal{D}} \alpha_j \leq c_S \quad \text{for all} \quad S \in \mathcal{S} \tag{39.10}$$

$$\alpha_j \geq 0 \quad \text{for all} \quad j \in \mathcal{D} \tag{39.11}$$

Intuitively, the dual variable α_j can be interpreted as the contribution of client j toward the total cost. Constraint (39.10) requires that for each star, the total contribution from the clients in the star should be no more than the cost of the star.

Note that if we raise the dual variables of all clients simultaneously and uniformly, the most cost-effective star will be the first star $S = (i, C)$ such that

$$\sum_{j \in S \cap \mathcal{D}} \alpha_j = c_S$$

or equivalently

$$\sum_{j \in C} \max(0, \alpha_j - c_{ij}) = f_i$$

On the basis of this, the MMSV algorithm can be restated as follows [18]:

The MMSV Algorithm: Dual-Ascent Representation.

1. At the beginning, all clients are *unconnected*, all facilities are *unopened*, and the *budget* of every client j, denoted by α_j, is initialized to 0. At every moment, each client j offers some money from its budget to each *unopened* facility i. The amount of this offer is equal to $\max(\alpha_j - c_{ij}, 0)$ if j is unconnected.

2. While there is an unconnected client, increase the budget of each *unconnected* client at the same rate, until one of the following events occurs:
   (a) For some unopened facility i, $\sum_{j \in U} \max(0, \alpha_j - c_{ij}) = f_i$. In this case, we open facility i, and for every unconnected client j with $\alpha_j \geq c_{ij}$, we connect j to i and remove it from U.
   (b) For some $j \in U$, and some facility i that is already open, $\alpha_j = c_{ij}$. In this case, we connect j to i and remove it from U.

The analysis of the MMSV algorithm can be sketched as follows. First of all, it is easy to see that the total cost of the solution produced by the algorithm is equal to $\sum_{j \in \mathcal{D}} \alpha_j$. Second, we show that there exists a constant $\gamma \geq 1$ such that

$$\sum_{j \in \mathcal{S}} \alpha_j \leq \gamma c_S \tag{39.12}$$

for any star $S \in \mathcal{S}$. Then from the dual problem (39.9)–(39.11) and by LP duality we can conclude that $\sum_{j \in \mathcal{D}} \alpha_j$ is no more than γ times the optimal value of the LP relaxation. Therefore, the algorithm is a γ-approximation algorithm. This idea is called dual-fitting.

Of course, we would like to find a γ such that Eq. (39.12) holds and γ is as small as possible. To do this, an elegant technique called the "factor revealing LP" is developed by Mahdian et al. [18]. They show that, for any star $S = (i, C)$, if the α_j's are ordered in such a way that $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_{|C|}$, then the following hold:

Lemma 39.1

For any $j \in C$, $\sum_{l=j}^{|C|} \max(\alpha_j - c_{jl}, 0) \leq f_i$.

Lemma 39.2

For any $1 \leq l < j \leq |C|, \alpha_j \leq \alpha_l + c_{il} + c_{ij}$.

The proof of Lemma 39.1 follows directly from the description of the algorithm, and Lemma 39.2 can be proven as follows [18]. Without loss of generality, we assume $\alpha_j > \alpha_l$ and let i' be the facility that client l is connected to by the MMSV algorithm. Thus, facility i' is open at time α_l. Therefore α_j cannot be greater than $c_{i'j}$ since otherwise j could be connected to facility i' at some time before α_j. Then using the above inequalities and the fact that the connection costs form a metric we have $\alpha_j \leq c_{i'j} \leq c_{i'l} + c_{il} + c_{ij} \leq \alpha_l + c_{il} + c_{ij}$.

Then it would be clear that if we define z_k as the optimal value of the following maximization problem for each $k \geq 1$:

$$\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{k} \alpha_i & \text{(LP1)}\\
\text{subject to} \quad & \forall\, 1 \leq i < k : \ \alpha_i \leq \alpha_{i+1} \\
& f + \sum_{j=1}^{k} d_j = 1 \\
& \forall\, 1 \leq i, j \leq k : \ \alpha_i \leq \alpha_j + d_i + d_j \\
& \forall\, 1 \leq i \leq k : \ \sum_{j=i}^{k} \max(\alpha_i - d_j, 0) \leq f \\
& \forall\, 1 \leq j \leq k : \ \alpha_j, d_j, f \geq 0
\end{aligned}$$

And let $\gamma := \sup_k\{z_k\}$. Then (39.12) is satisfied and it can be shown that $\gamma \leq 1.861$. And this implies that the MMSV algorithm is a 1.861 algorithm.

The linear program (LP1) is called a factor-revealing LP. As we have seen, it is used to derive an upper bound for the performance guarantee of the MMSV algorithm. Mahdian et al. [18] also show that the factor-revealing LP is helpful to construct the worst-case instances. Indeed, they show that the performance guarantee of the algorithm is lower-bounded by 1.81.

39.3 BiFactor Approximation

In this section, we first present the second algorithm in Jain et al. [17], which was first presented in Jain et al. [19]. We called it the JMS algorithm. We also present the 1.52-approximation algorithm due to Mahdian et al. [20]. The analysis of the JMS algorithm has the feature that allows the approximation factor for the facility cost to be different from the approximation factor for the connection cost, and gives a way to compute the trade-off between these two factors. The following definition captures this notion [22]:

Definition 39.1

An algorithm is called a (γ_f, γ_c)-approximation algorithm *for a facility location problem, if for every instance* $\mathcal{I}$ *of the problem, and for every solution SOL for* $\mathcal{I}$ *with facility cost* F_{SOL} *and connection cost* C_{SOL}*, the cost of the solution found by the algorithm is at most* $\gamma_f F_{SOL} + \gamma_c C_{SOL}$.

39.3.1 The 1.61-Approximation Algorithm

Now we present the JMS algorithm, which is very similar to the MMSV algorithm. In the MMSV algorithm, once a client j is connected to a facility i, α_j will be fixed and j will not be switched to other facilities. However, it is possible that later on, when the algorithm opens a new facility, say i', the connection cost $c_{i'j}$ is less than c_{ij}. Therefore, j can be switched to facility i' and it will save $c_{ij} - c_{i'j}$, which can be used to open facility i'.

Therefore, one could modify the definition of the cost-effectiveness of a star $S = (i, C)$ as follows. Let U be the set of unconnected clients. And for each $j \in \mathcal{D} \setminus U$, let $\sigma(j)$ be the facility that j is connected to. Then the cost-effectiveness of a star $S = (i, C)$ with $C \subseteq U$ is defined as

$$\frac{f_i - \sum_{j \in \mathcal{D} \setminus U} \max(0, c_{ij} - c_{\sigma(j)j}) + \sum_{j \in C} c_{ij}}{|C|}$$

Essentially, the JMS algorithm is the MMSV algorithm with the modified definition. Again, for the purpose of analyzing the algorithm, we state the JMS algorithm as a dual-ascent algorithm, as it has been done in Ref. [19].

The JMS Algorithm.

1. At the beginning, all clients are *unconnected*, all facilities are *unopened*, and the *budget* of every client j, denoted by B_j, is initialized to 0. At every moment, each client j offers some money from its budget to each *unopened* facility i. The amount of this offer is equal to $\max(B_j - c_{ij}, 0)$ if j is unconnected, and $\max(c_{i'j} - c_{ij}, 0)$ if it is connected to some other facility i'.
2. While there is an unconnected client, increase the budget of each *unconnected* client at the same rate, until one of the following events occurs:
   (a) For some unopened facility i, the total offer that it receives from clients is equal to the cost of opening i. In this case, we open facility i, and for every client j (connected or unconnected), which has a nonzero offer to i, we connect j to i.
   (b) For some unconnected client j, and some facility i that is already open, the budget of j is equal to the connection cost c_{ij}. In this case, we connect j to i.

One may notice that the JMS algorithm finds a solution in which there is no unopened facility that one can open to decrease the cost (without closing any other facility). This is because for each client j and facility i, j offers to i the amount that it would save in the connection cost if it gets its service from i.

The analysis of the algorithm also relies on dual-fitting and factor-revealing LP.

Again, the total cost of the solution produced by the algorithm is equal to $\sum_{j\in\mathcal{D}}\alpha_j$. Also, from the dual problem (39.5)–(39.8) we can show that, if there exists γ_f and γ_c such that

$$\sum_{j\in\mathcal{S}}\alpha_j \leq \gamma_f f_i + \gamma_c \sum_{j\in C} c_{ij} \tag{39.13}$$

for any star $S = (i, C) \in \mathcal{S}$, then by LP duality the algorithm is a (γ_f, γ_c)-approximation algorithm.

To develop a factor-revealing LP to find such γ_f and γ_c, we would like to have some results analogous to Lemmas 39.1 and 39.2. Again, for any star $S = (i, C)$, we assume the α_j's are ordered in such a way that $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_{|C|}$. We need more variables to capture the execution of the JMS algorithm. For every $l < j$, if client l is connected to some facility i at time α_j, let $r_{lj} = c_{il}$; otherwise, let $r_{lj} = \alpha_l$ (this case will occur only if $\alpha_l = \alpha_j$). Then one can prove the following lemmas:

Lemma 39.3

For any $l \in C$, $r_{l,l+1} \geq r_{l,l+2} \geq \cdots, r_{l,|C|}$.

Lemma 39.4

For any $j \in C$, $\sum_{l=1}^{j-1}\max(\alpha_{lj} - c_{il}, 0) + \sum_{l=j}^{|C|}\max(\alpha_j - c_{jl}, 0) \leq f_i$.

Lemma 39.5

For any $1 \leq l < j \leq |C|$, $\alpha_j \leq r_{lj} + c_{il} + c_{ij}$.

Using the above lemmas, we can then establish the following result due to Ref. [19]:

Theorem 39.1

Let $\gamma_f \geq 1$ *be fixed and* $\gamma_c := \sup_k\{z_k\}$, *where* z_k *is the solution of the following optimization program which is referred to as the factor-revealing LP.*

$$\text{maximize} \quad \frac{\sum_{i=1}^{k}\alpha_i - \gamma_f f}{\sum_{i=1}^{k} d_i} \tag{LP2}$$

$$\text{subject to} \quad \forall\, 1 \leq i < k:\ \alpha_i \leq \alpha_{i+1} \tag{39.14}$$

$$\forall\, 1 \leq j < i < k:\ r_{j,i} \geq r_{j,i+1} \tag{39.15}$$

$$\forall\, 1 \leq j < i \leq k:\ \alpha_i \leq r_{j,i} + d_i + d_j \tag{39.16}$$

$$\forall\, 1 \leq i \leq k:\ \sum_{j=1}^{i-1}\max(r_{j,i} - d_j, 0) + \sum_{j=i}^{k}\max(\alpha_i - d_j, 0) \leq f \tag{39.17}$$

$$\forall\, 1 \leq j \leq i \leq k:\ \alpha_j, d_j, f, r_{j,i} \geq 0 \tag{39.18}$$

Then the JMS algorithm is a (γ_f, γ_c)*-approximation algorithm for UFLP.*

Jain et al. [19] have also shown that for $\gamma_f = 1$ we have $\gamma_c \leq 2$. Mahdian et al. [20] proved that $\gamma_c \leq 1.78$ when $\gamma_f = 1.11$ and they used this pair of ratios to get the 1.52-approximation algorithm. We call their algorithm the MYZ algorithm. The MYZ algorithm also uses the idea of cost scaling, which was originally proposed by Charikar and Guha [9].

39.3.2 The 1.52-Approximation Algorithm

The MYZ algorithm has two phases.

Phase I. We scale up the opening costs of all facilities by a factor of δ (which is a constant that will be fixed later) and then run the JMS algorithm to find a solution.

Phase II. In the *second* phase of the algorithm, we decrease the scaling factor δ at rate 1, so at time t, the cost of facility i has reduced to $(\delta - t) f_i$. If at any point during this process, a facility could be opened without increasing the total cost (i.e., if the opening cost of the facility equals the total amount that clients can save by switching their "service provider" to that facility), then we open the facility and connect each client to its closest open facility. We stop when the scaling factor becomes 1.

Intuitively, the facilities that are opened in Phase I are those that are very economical, because we weigh the facility cost more than the connection cost in the objective function.

We remark that Phase II is equivalent to a greedy procedure introduced by Guha and Khuller [2] and Charikar and Guha [9]. In this procedure, in each iteration, we pick a facility u of opening cost f_u such that if by opening u, the total connection cost decreases from C to C'_u, and the ratio $(C - C'_u - f_u)/f_u$ is maximized. If this ratio is positive, then we open the facility u, and iterate; otherwise we stop. In Phase II of the MYZ algorithm, the first facility u that is opened corresponds to the minimum value of t, or the maximum value of $\delta - t$, for which we have $(\delta - t) f_u = C - C'_u$. In other words, our algorithm picks the facility u for which the value of $(C - C'_u)/f_u$ is maximized, and stops when this value becomes less than or equal to 1 for all u. This is the same as what the Charikar–Guha–Khuller procedure does.

Mahdian et al. [20] proved that the JMS algorithm is a (1.11, 1.78)-approximation algorithm for the metric UFLP. It follows directly from the results of Refs. [2,9]—which is essentially the same as the one proved in Theorem 39.3 below—that the above algorithm is a 1.52-approximation algorithm. However, Mahdian et al. [22] were able to show that the performance guarantee of the MYZ algorithm can be bounded by analyzing a *single* factor-revealing LP. Their analysis depends on a modified implementation of Phase II as the following.

Instead of decreasing the scaling factor continuously from δ to 1, we decrease it discretely in L steps, where L is a constant. Let δ_i denote the value of the scaling factor in the ith step. Therefore, $\delta = \delta_1 > \delta_2 > \cdots > \delta_L = 1$. We will fix the value of the δ_i's later. After decreasing the scaling factor from δ_{i-1} to δ_i, we consider facilities in an *arbitrary* order, and open those that can be opened without increasing the total cost. We denote this modified algorithm by MYZ_L. Clearly, if L is sufficiently large (depending on the instance), the algorithm MYZ_L computes the same solution as Algorithm MYZ.

Then, Mahdian et al. [20] were able to prove that

Theorem 39.2

Let (ξ_f, ξ_c) be such that $\xi_f \geq 1$ and ξ_c is an upper bound on the solution of the following maximization program for every k:

$$\text{maximize} \quad \frac{\sum_{j=1}^{k}\left(\frac{\alpha_j}{\delta} + \sum_{i=1}^{L-1}\left(\frac{1}{\delta_{i+1}} - \frac{1}{\delta_i}\right) r_{j,k+i}\right) - \xi_f f}{\sum_{i=1}^{k} d_i} \tag{LP2}$$

$$\text{subject to} \quad \forall\, 1 \leq i < k:\ \alpha_i \leq \alpha_{i+1} \tag{39.19}$$

$$\forall\, 1 \leq j < i < k:\ r_{j,i} \geq r_{j,i+1} \tag{39.20}$$

$$\forall\, 1 \leq j < i \leq k:\ \alpha_i \leq r_{j,i} + d_i + d_j \tag{39.21}$$

$$\forall\, 1 \leq i \leq k:\ \sum_{j=1}^{i-1} \max(r_{j,i} - d_j, 0) + \sum_{j=i}^{k} \max(\alpha_i - d_j, 0) \leq \delta f \tag{39.22}$$

$$\forall\, 1 \leq i \leq L:\ \sum_{j=1}^{k} \max(r_{j,k+i} - d_j, 0) \leq \delta_i f \tag{39.23}$$

$$\forall\, 1 \leq j \leq i \leq k:\ \alpha_j, d_j, f, r_{j,i} \geq 0 \tag{39.24}$$

Then, algorithm MYZ_L is a (ξ_f, ξ_c)-approximation algorithm for UFLP.

The following theorem follows from the above factor-revealing LP, and it immediately implies that the metric LP can be approximated by a factor of 1.52 by choosing $\delta = 1.504$.

Theorem 39.3

Let (γ_f, γ_c) be a pair given by the maximization program (LP2) in Theorem 39.1, and $\delta \geq 1$ be an arbitrary number. Then for every ϵ, if L is a sufficiently large constant, algorithm A_L is a $(\gamma_f + \ln(\delta) + \epsilon, 1 + \frac{\gamma_c - 1}{\delta})$-approximation algorithm for the metric UFLP.

We note that although the performance guarantee of the MYZ algorithm can be upper-bounded by analyzing a single factor-revealing LP, it is not clear whether the bound provided by the factor-revealing LP is tight or not. Detailed discussion of the MYZ algorithm can be found in [32], which is the journal version of Refs. [20,22].

39.4 Quasi-Greedy Algorithm for Two-Level Facility Location

In this section, we consider a generalization of the metric UFLP, called the k-level uncapacitated facility location problem (k-LFLP). In this problem, the demands must be routed among facilities in a hierarchical order, that is, from the highest level (the factories) down to the lowest (the retailers), before reaching the clients. The k-LFLP arises naturally in designing logistic systems.

The k-LFLP can be formulated formally as follows. We are given a set of clients $\mathcal{D}$ and k-level sets of facilities $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k$. Denote $\mathcal{P} = \mathcal{F}_1 \times \mathcal{F}_2 \times \cdots \times \mathcal{F}_k$ and $\mathcal{F} = \cup_{t=1}^{k} \mathcal{F}_t$. Each client $j \in D$ must be served by an open path $p = (i_1, i_2, \ldots, i_k) \in \mathcal{P}$ of k facilities with exactly one from each of the k levels, where a path p is open if and only if every facility on the path is open. There is a facility cost f_{i_t} for opening facility $i_t \in \mathcal{F}_t$ $(1 \leq t \leq k)$. Furthermore, if client $j \in \mathcal{D}$ is served by an open path $p = (i_1, i_2, \ldots, i_k) \in \mathcal{P}$ a connection cost c_{jp} is incurred where $c_{jp} = c_{ji_1} + \sum_{t=2}^{k} c_{i_{t-1}i_t}$ and c_{ji} is the connection cost between j and i for $j, i \in \mathcal{D} \cup \mathcal{F}$. Here, we wish to open a subset of facilities such that each client is assigned to an open path and the total cost is minimized, that is, to choose $\emptyset \neq S_t \subset \mathcal{F}_t$, $t = 1, 2, \ldots, k$, such that

$$\sum_{j \in D} \min_{p \in S_1 \times S_2 \times \cdots \times S_k} c_{jp} + \sum_{t=1}^{k} \sum_{i_t \in S_t} f_{i_t}$$

is minimized. We also assume that the connection costs are nonnegative, symmetric, and satisfy the triangle inequality, that is, for each $i, j, l \in \mathcal{D} \cup \mathcal{F}$, $c_{ij} \geq 0$, $c_{ij} = c_{ji}$ and $c_{ij} \leq c_{il} + c_{lj}$. The metric UFLP is just 1-LFLP.

The 2-LFLP is the most studied special case of k-LFLP in the literature of *Operations Research* for $k \geq 2$ [23]. Although it is the simplest model among all the k-LFLP for $k \geq 2$, the 2-LFLP has some fundamental structural differences from the 1-LFLP. For example, the 2-LFLP does not possess the so-called supermodularity, a well-known property for the 1-LFLP [24]. This property is often helpful in designing branch-and-cut algorithms and in analyzing some approximation algorithms. Thus, the 2-LFLP needs new techniques.

One can easily see that the lower bound 1.463 of the 1-LFLP also applies to the 2-LFLP, and no better lower bound is known. In Ref. [5], the algorithm for the 1-LFLP has been extended to the 2-LFLP with an approximation ratio 3.16. Later on, Aardal et al. [25] showed that the k-LFLP can be approximated in polynomial time by a factor of 3 for any $k \geq 2$ using an LP relaxation. However, their algorithm does not possess a better performance guarantee for $k = 2$, and neither do a series of recently proposed faster combinatorial algorithms due to Meyerson et al. [26], Guha et al. [27], Bumb and Kern [28], and Ageev [29]. In fact, the algorithms of Refs. [25,28,29] will produce solutions whose open paths are disjoint, and Edwards [30] showed that such algorithms cannot have worst-case ratios that are better than 3 even for $k = 2$. Ageev et al. [31] proposed two different combinatorial algorithms and showed that "the better of the two" has a performance guarantee 2.43, although each of them has a performance guarantee of at least 3.

In this section, we present the algorithm developed by Zhang [21]. The main result of Ref. [21] is that the 2-LFLP can be approximated in polynomial time by a factor of 1.77, which achieves a significant improvement over previous results. The improved ratio is achieved by using what we call *a quasi-greedy approach.* Our algorithm is analyzed by using the technique of factor-revealing LP presented in previous sections.

To design a greedy algorithm for the 2-LFLP, we should define the candidate set and the greedy function. It is known that the open paths of an optimal solution of the 2-LFLP form a forest [30]. Thus a natural choice of the set of candidates would be the set of all trees, where each tree $T = (i, S_i, D_i)$ consists of one facility $i \in \mathcal{F}_2$, a set of first-level facilities $S_i \subset \mathcal{F}_1$, and a set of clients $D_i \in \mathcal{D}$. Then we can define the cost-effectiveness of a tree. The cost associated with a tree $T = (i, S_i, D_i)$ is

$$f_i + \sum_{k \in S_i} f_k + \sum_{j \in D_i} \min_{k \in S_i} c_{jki}$$

and the cost-effectiveness of this tree is

$$\frac{f_i + \sum_{k \in S_i} f_k + \sum_{j \in D_i} \min_{k \in S_i} c_{jki}}{|D_i|}$$

On the basis of this definition, we can design a greedy algorithm for the 2-LFLP similar to the MMSV algorithm. However, it should be noted that the problem of choosing a tree with minimum cost-effectiveness itself is at least as hard as the metric UFLP. Therefore, at each step, we do not choose the best candidate. Instead, we use an approximation algorithm to find a "good" candidate. The resulting algorithm is called a quasi-greedy algorithm [21].

Similarly, we can also modify the definition of the cost-effectiveness as it has been done in the JMS algorithm. The quasi-greedy algorithm developed in Ref. [21], denoted by QG, is a dual-ascent representation of the greedy algorithm with a modified definition of the cost-effectiveness. As we have mentioned earlier, at each step of the algorithm, we need to solve another NP-hard problem. In fact, this NP-hard problem is called the maximization version of the facility location problem, denoted by Max-1-LFLP.

The Max-1-LFLP is defined as follows. We are given a set of client $\mathcal{D}$ and a set of facilities $\mathcal{F}$. The facility cost for opening facility i is f_i and the revenue generated by assigning client j to facility i is $d_{ij} \geq 0$. Here, the d_{ij}'s may not satisfy the triangle inequality. The objective is to open a subset of the facilities of $\mathcal{F}$ and then assign each of the clients in $\mathcal{D}$ to an open facility such that the net profit is maximized. The Max-1-LFLP can be formulated as the following integer program:

$$\begin{aligned} \text{Max} \quad & \sum_{i \in \mathcal{F}, j \in \mathcal{D}} d_{ij} x_{ij} - \sum_{i \in \mathcal{F}} f_i y_i \\ \text{s.t.} \quad & \sum_{i \in \mathcal{F}} x_{ij} \leq 1 \quad \text{for all} \quad j \in \mathcal{D} \\ & x_{ij} \leq y_i \quad \text{for all} \quad i \in \mathcal{F}, j \in \mathcal{D} \\ & x_{ij}, y_i \in \{0, 1\} \quad \text{for all} \quad i \in \mathcal{F}, j \in \mathcal{D} \end{aligned} \tag{39.25}$$

where $y_i = 1$ if we decide to open facility i, otherwise $y_i = 0$; $x_{ij} = 1$ if client j is assigned to facility i, otherwise $x_{ij} = 0$.

The Max-1-LFLP and the minimization 1-LFLP are equivalent from the perspective of optimization, but not from that of approximation. Approximation algorithms for the Max-1-LFLP have a longer history than those for the 1-LFLP [33,34]. However, the results of Refs. [33,34] do not help in establishing our result for the 2-LFLP. Instead, the following simple LP-rounding algorithm has been used to solve the Max-1-LFLP in Ref. [21].

Algorithm MAX

Step 1. Solve the following LP and obtain an optimal solution (x, y):

$$\begin{aligned} \text{Max} \quad & (1 - \frac{1}{e}) \cdot \sum_{i \in \mathcal{F}, j \in \mathcal{D}} d_{ij} x_{ij} - \sum_{i \in \mathcal{F}} f_i y_i \\ \text{s.t.} \quad & \sum_{i \in \mathcal{F}} x_{ij} \leq 1 \quad \text{for all} \quad j \in \mathcal{D} \\ & x_{ij} \leq y_i \quad \text{for all} \quad i \in \mathcal{F}, j \in \mathcal{D} \\ & 0 \leq x_{ij}, y_i \leq 1 \quad \text{for all} \quad i \in \mathcal{F}, j \in \mathcal{D} \end{aligned} \tag{39.26}$$

Step 2. For each $i \in \mathcal{F}$, open facility i independently with probability y_i, and assign each client $j \in \mathcal{D}$ to an open facility with maximum revenue. Let the resulting solution be $(\hat{x}, \hat{y})$ and the corresponding facility cost and revenue be F and C, respectively.

Consider any feasible solution, whose total profit is assumed to be $C^* - F^*$, where C^* and F^* correspond to the total revenue and the total facility cost, respectively. It can be shown that Algorithm MAX produces a solution with profit $C - F \geq (1 - \frac{1}{e})C^* - F^*$.

Then we are ready to present the quasi-greedy algorithm for the 2-LFLP.

The Algorithm QG

1. We introduce a notion of time. The algorithm starts at time 0. At this time, all clients are unconnected and all facilities are unopen. Let U be the set of unconnected clients. Thus at this time $U = \mathcal{D}$. And $\alpha_j = 0$ for each $j \in \mathcal{D}$.

   At each moment, every client j will have some money B_j available to offer to each unopen facility in $\mathcal{F}_2$, where $B_j = \alpha_j$ if j is unconnected, and $B_j = c_{jp}$ if j is currently connected to an open path p. The amount of offers received by a facility $i \in \mathcal{F}_2$ is computed as follows. Consider any $i \in \mathcal{F}_2$. For each $j \in \mathcal{D}$ and $k \in \mathcal{F}_1$, define $d_{kj} = \max\{B_j - c_{jki}, 0\}$, where $c_{jki} = c_{jk} + c_{ki}$, and $\hat{f}_k = 0$ if $k \in \mathcal{F}_1$ is already open; $\hat{f}_k = f_k$ otherwise. Then we obtain an instance of the Max-1-LFLP. We solve this instance by Algorithm MAX and obtain a feasible solution. This profit (could be negative) is the amount of offers received by the facility i. Note that each client j can make an offer to a facility $i \in \mathcal{F}_2$ through exactly one facility $\sigma_i(j) \in \mathcal{F}_1$, where $\sigma_i(j)$ is the facility to which j is assigned by Algorithm MAX. The contribution made by client j to facility i is equal to $d_{\sigma_i(j)j}$.
2. While $U \neq \emptyset$, increase the time and simultaneously increase α_j at the same rate for each $j \in U$, until one of the following events occurs:
   (a) For an unopen facility $i \in \mathcal{F}_2$, the total amount of offers that it has received from the clients is equal to f_i. In this case, we open facility i. And for each client $j \in \mathcal{D}$ who has made nonzero contribution to i, we open facility $\sigma_i(j) \in \mathcal{F}_1$ and assign j to the path $(\sigma_i(j), i)$. Furthermore, if $j \in U$, then remove j from U (and stop increasing α_j).
   (b) For a client $j \in U$ and open facilities $i \in \mathcal{F}_2$ and $k \in \mathcal{F}_1$, $\alpha_j = c_{jki}$, then assign j to the path (k, i) and remove j from U (and stop increasing α_j).

To implement Algorithm QG in polynomial time, we notice that the total number of possible events is bounded by $|\mathcal{D}| + |\mathcal{F}_2|$. At any time, we need to find the minimum value of how much the α_j's should increase such that the next event will occur. This can be done in polynomial time (but not strongly polynomial time) by performing a bisection search. Another way for implementing the algorithm is that we discretize the time and only consider the values of α_j's that are powers of $(1 + \epsilon)$, that is, $\{0, 1, (1 + \epsilon), (1 + \epsilon)^2, (1 + \epsilon)^3, \ldots, \}$ for any given constant $\epsilon > 0$. Therefore, the algorithm can be implemented in polynomial time for any given constant $\epsilon > 0$.

Again, it is easy to show that the total cost of the solution produced by Algorithm QG is $\sum_{j \in \mathcal{D}} \alpha_j$. If we could prove that, for any tree $T = (i, S_i, D_i)$,

$$\sum_{j \in D_i} \alpha_j \leq R_f \left(f_i + \sum_{k \in S_i} f_k \right) + R_c \sum_{j \in D_i} \min_{k \in S_i} c_{jki}$$

then Algorithm QG must be a (R_f, R_c)-approximation algorithm for the 2-LFLP. This is done by constructing a factor-revealing LP.

Now we consider a tree $T = (i, S_i, D_i)$. We assume that $|D_i| = n$ and let $m_j = \min_{k \in S_i} c_{jki}$. Furthermore, without losing generality, we assume that $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_n$.

For each $j : 1 \leq j \leq n$, consider the situation of the algorithm at time $t = \alpha_j$. For each $l \leq j - 1$, if l is connected to a path p before time t (i.e., l was connected to a path at time $\alpha_j/(1 + \epsilon)$, then let $r_{l,j} = c_{lp}$; otherwise, let $r_{l,j} = \alpha_l$. In the latter case, $\alpha_l = \alpha_j$.

Then we can show some results analogous to Lemmas 39.3–39.5.

Lemma 39.6

For each $1 \leq l < j \leq n$*, we have* $r_{l,j} \geq r_{l,j+1}$.

Lemma 39.7

For any $j : 1 \leq j \leq n$*,*

$$\sum_{l=1}^{j-1} \max\left(r_{l,j}/(1+\epsilon) - m_l, 0\right) + \sum_{l=j}^{n} \max\left(\alpha_j/(1+\epsilon) - m_l, 0\right) \leq \frac{e}{e-1}\left(f_i + \sum_{k \in S_i} f_k\right)$$

Lemma 39.8

For any $1 \leq l < j \leq n, \alpha_j/(1+\epsilon) \leq r_{l,j} + m_j + m_l$

Notice that, if we had been able to solve the Max-1-LFLP optimally, then Lemma 39.1 would be the following: For any $j : 1 \leq j \leq n$,

$$\sum_{l=1}^{j-1} \max\left(r_{l,j}/(1+\epsilon) - m_l, 0\right) + \sum_{l=j}^{n} \max\left(\alpha_j/(1+\epsilon) - m_l, 0\right) \leq \left(f_i + \sum_{k \in S_i} f_k\right)$$

Then we would get a 1.52-approximation algorithm for the 2-LFLP.

Comparing Lemmas 39.6, 39.7, and 39.8 with Lemmas 39.3, 39.4, and 39.5, respectively, we see that if (γ_f, γ_c) satisfies Theorem 39.1, then Algorithm QG is a $(\frac{e}{e-1}\gamma_f, \gamma_c)$-approximation algorithm. In particular, we can choose $\gamma_f = 1.118$ and $\gamma_c = 1.77$, from which it will follow that Algorithm QG is a 1.77-approximation algorithm.

References

[1] Balinski, M. L., On finding integer solutions to linear programs, *Proc. IBM Sci. Comput. Symp. on Combinatorial Problems,* 1966, p. 225.

[2] Guha, S. and Khuller, S., Greedy strikes back: improved facility location algorithms, *J. Algorithms,* 31, 228, 1999.

[3] Chudak, F. A. and Williamson, D., improved approximation algorithms for capacitated facility location problems, *Proc. 7th Conf. Integer Programming and Combinatorial Optimization*, 1999, p. 99.

[4] Hochbaum, D. S., Heuristics for the fixed cost median problem, *Math. Prog.*, 22, 148, 1982.

[5] Shmoys, D. B., Tardos, E., and Aardal, K. I., Approximation algorithms for facility location problems, *Proc. of STOC*, 1997, p. 265.

[6] Chudak, F. A. and Shmoys, D. B., Improved approximation algorithms for the uncapacitated facility location problem, *SIAM J. Comput.*, 33, 1, 2003.

[7] Sviridenko, M., An 1.582-approximation algorithm for the metric uncapacitated facility location problem, *Proc. 9th Conf. on Integer Programming and Combinatorial Optimization*, 2002, p. 240.

[8] Korupolu, M. R., Plaxton, C. G., and Rajaraman, R., Analysis of a local search heuristic for facility location problems, *Proc. of SODA,* 1998, p. 1.

[9] Charikar, M. and Guha, S., Improved combinatorial algorithms for facility location and k-median problems, *Proc. of FOCS,* 1999, p. 378.

[10] Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala K., and Pandit, V., Local search heuristic for k-median and facility location problems, *Proc. of STOC,* 2001, p. 21.

[11] Pál, M., Tardos, É., and Wexler, T., Facility location with hard capacities, *Proc. of FOCS,* 2001, p. 329.

[12] Mahdian, M. and Pál, M., Universal facility location, *Proc. 11th Ann. European Symp. on Algorithms*, 2003, p. 409.

[13] Zhang, J., Chen, B., and Ye, Y., A multi-exchange local search algorithm for the capacitated facility location problem, *Math. Oper. Res.*, 30(2), 389, 2005.

[14] Jain, K. and Vazirani, V. V., Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation, *JACM*, 48, 274, 2001.

[15] Mettu, R. R. and Plaxton, C. G., The online median problem, *SIAM J. Comput.*, 32, 816, 2003.

[16] Thorup, M., Quick k-median, k-center, and facility location for sparse graphs, in *Proc of ICALP*, Orejas, F., Spirakis, P. G., and van Leeuwen, J., Eds., Lecture Notes in Computer Science, Vols. 2076, 260, Springer, Berlin, 2001, p. 249.

[17] Jain, K., Mahdian, M., Markakis, E., Saberi, A., and Vazirani, V. V., Approximation algorithms for facility location via dual fitting with factor-revealing LP, *JACM*, 50, 795, 2003.

[18] Mahdian, M., Markakis, E., Saberi, A., and Vazirani, V. V., A greedy facility location algorithm analyzed using dual fitting, in *Proc. 5th Int. Workshop on Randomization and Approximation Techniques in Computer Science*, Goemans, M. X., Jansen, K., Rolim, J. D. P., and Trevisan, L., Eds., Lecture Notes in Computer Science, Vol. 2129, Springer, Berlin, 2001, p. 127.

[19] Jain, K., Mahdian, M., and Saberi, A., A new greedy approach for facility location problems, *Proc. of STOC*, 2002, p. 731.

[20] Mahdian, M., Ye, Y., and Zhang, J., Improved approximation algorithms for metric facility location problems, *Proc. 5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization*, Lecture Notes in Computer Science, Vol. 2462, Springer, Berlin, 2002, p. 229.

[21] Zhang, J., Approximating the two-level facility location problem via a quasi-greedy approach, *Math. Prog.*, 108(1), 159, 2006.

[22] Mahdian, M., Ye, Y., and Zhang, J., A 2-approximation algorithm for the soft-capacitated facility location problem, *Proc. 6th Int. Workshop on Approximation Algorithms for Combinatorial Optimization*, Lecture Notes in Computer Science, Vol. 2764, Springer, Berlin, 2003, p. 129.

[23] Aardal, K., Labbe, M., Leung, J., and Queyranne, M., On the two-level uncapacitated facility location problem, *INFORMS J. Computing*, 8, 289, 1996.

[24] Labbe, M., The multi-level uncapacitated facility location problem is not submodular, *Eur. J. Oper. Res.*, 72, 607, 1996.

[25] Aardal, K., Chudak, F. A., and Shmoys, D. B., A 3-approximation algorithm for the k-Level uncapacitated facility location problem, *Inf. Process. Lett.* 72, 161, 1999.

[26] Meyerson, A., Munagala, K., and Plotkin, S., Cost-distance: two-metric network design, *Proc. of FOCS*, 2000, p. 624.

[27] Guha, S., Meyerson, A., and Munagala, K., Hierarchical placement and network design problems, *Proc. of FOCS*, 2000, p. 603.

[28] Bumb, A. F. and Kern, W., A simple dual ascent algorithm for the multilevel facility location problem, *Proc. 4th Int. Workshop on Approximation Algorithms for Combinatorial Optimization*, Lecture Notes in Computer Science, Vol. 2129, Springer, Berlin, 2001, p. 55.

[29] Ageev, A., Improved approximation algorithms for multilevel facility location problems, *Oper. Res. Lett.* 30, 327, 2002.

[30] Edwards, N., Approximation Algorithms for the Multi-level Facility Location Problem, Ph.D. thesis, School of Operations Research and Industrial Engineering, Cornell University, 2001.

[31] Ageev, A., Ye, Y., and Zhang, J., Improved combinatorial approximation algorithms for the k-level facility location problem, *SIAM J. Discrete Math.*, 18(1), 207, 2004.

[32] Mahdian, M., Ye, Y., and Zhang, J., Approximation algorithms for metric facility location problems, *SIAM Journal on Computing*, 36(2), 411, 2006.

[33] Cornuéjols, G., Fisher, M. L., and Nemhauser, G. L., Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms, *Manage. Sci.*, 23, 789, 1977.

[34] Ageev, A. and Sviridenko, M., An 0.828-approximation algorithm for uncapacitated facility location problem, *Discrete Appl. Math.*, 93, 289, 1999.

40

Prize-Collecting Traveling Salesman and Related Problems

Giorgio Ausiello
University of Rome "La Sapienza"

Vincenzo Bonifaci
University of Rome "La Sapienza"

Stefano Leonardi
University of Rome "La Sapienza"

Alberto Marchetti-Spaccamela
University of Rome "La Sapienza"

40.1 Introduction

The most general version of the Prize-Collecting Traveling Salesman Problem (PCTSP) was first introduced by Balas [1]. In this problem, a salesman has to collect a certain amount of prizes (the *quota*) by visiting cities. A known prize can be collected in every city. Furthermore, by not visiting a city, the salesman incurs a pecuniary *penalty*. The goal is to minimize the total travel distance plus the total penalty, while starting from a given city and collecting the quota.

The problem generalizes both the Quota TSP, which is obtained when all the penalties are set to zero, and the Penalty TSP (PTSP) (sometimes unfortunately also called PCTSP), in which there is no required quota, only penalties. A special case of the Quota TSP is the k-TSP, in which all prizes are unitary (k is the quota). The k-TSP is strongly tied to the problem of finding a tree of minimum cost spanning any k vertices in a graph, called the k-Minimum Spanning Tree (k-MST) problem.

The k-MST and the k-TSP are NP-hard. They have been the subject of several studies for good approximation algorithms [2–6]. A 2-approximation scheme for both the k-MST and the k-TSP given by Garg [6] is the best known approximation ratio. Interestingly enough, all these algorithms use the primal-dual algorithm of Goemans and Williamson [7] for the Prize-Collecting Steiner Tree as a subroutine.

The Quota TSP was also considered by some researchers. It was considered by Awerbuch et al. [8], who gave an $O(\log^2(\min(Q, n)))$ approximation algorithm for instances with n cities and quota Q. Ausiello et al. [9] give an algorithm with an approximation ratio of 5 for what could be called the Quota MST, by extending some of the ideas of Ref. [5].

The PTSP has apparently a better approximation ratio. Goemans and Williamson [7], as an application of their primal-dual technique, give an approximation algorithm with a ratio of 2 for the PTSP. Their algorithm uses a reduction to the Prize-Collecting Steiner Tree Problem. The running time of the algorithm was reduced from $O(n^2 \log n)$ to $O(n^2)$ by Gabow and Pettie [10].

We finally remark that both the MST and the TSP also admit a *budget* version; in these cases a budget B is specified in the input and the objective is to find the largest k-MST, respectively the k-TSP, whose cost is no more than the given budget.

Awerbuch et al. [8] in the aforementioned work, were the first to give an approximation algorithm for the general PCTSP. Their approximation ratio is again $O(\log^2(\min(Q, n)))$. They achieve this ratio by concatenating the tour found by the Quota TSP algorithm to a tour found by the Goemans–Williamson algorithm. As an application of the 5-approximate algorithm for the Quota MST [9] it follows that the approximation ratio of the PCTSP is constant.

In what follows, we introduce formal details and we provide a review of the main results in the area. In Section 40.2, we present the algorithm by Goemans and Williamson for the Prize-Collecting Steiner Tree, which is a basic building block for all the other algorithms in this chapter and has also an application to the PTSP. In Section 40.3, we analyze Garg's technique showing a 5-approximation for both the k-MST and k-TSP and show how to extend it to the Quota TSP. In Section 40.4, we describe an algorithm for the general PCTSP that builds over the algorithms for Quota TSP and PTSP. In Section 40.5, we show some applications of these algorithms to the minimum latency problem and graph searching.

40.2 The Prize-Collecting Steiner Tree Problem and Penalty Traveling Salesman Problem

40.2.1 Definitions

Prize-Collecting Steiner Tree. Given an undirected graph $G = (V, E)$ with vertex penalties $\pi : V \to \mathbb{Q}^+$, edge costs $c : E \to \mathbb{Q}^+$ and a root node r, the Prize-Collecting Steiner Tree problem asks to find a tree $T = (V_T, E_T)$ including r that minimizes

$$c(T) = \sum_{e \in E_T} c_e + \sum_{v \in V \setminus V_T} \pi_v$$

Penalty Traveling Salesman Problem. Given an undirected graph $G = (V, E)$ with vertex penalties $\pi : V \to \mathbb{Q}^+$, edge costs $c : E \to \mathbb{Q}^+$ satisfying the triangle inequality and a root node r, the PTSP asks to find a tour $T = (V_T, E_T)$ including r that minimizes

$$c(T) = \sum_{e \in E_T} c_e + \sum_{v \in V \setminus V_T} \pi_v$$

40.2.2 History of the Results

The first approximation algorithms for the Prize-Collecting Steiner Tree and PTSP were developed by Bienstock et al. [11]. They gave LP-based algorithms achieving a 3-approximation for the PCST and a 5/2-approximation for the PTSP with triangle inequality. Both bounds were later improved to 2 by Goemans and Williamson [7] with a combinatorial algorithm. The NP-hardness (more precisely, APX-hardness) of the problems follows from that of the Steiner Tree problem [12] and the TSP [13], respectively.

40.2.3 The Primal-Dual Algorithm of Goemans and Williamson

We review the algorithm of Goemans and Williamson [7] for the *Prize-Collecting Steiner Tree* and *PTSP*. We are given an undirected graph $G = (V, E)$, nonnegative edge costs c_e, and nonnegative vertex penalties π_i. The goal in the Prize-Collecting Steiner Tree problem is to minimize the total cost of a Steiner tree and the penalties of the vertices that are not spanned by the Steiner tree. In the PTSP, we aim to minimize the cost of a tour and of the penalties of the vertices that are not included in the tour. In this section we revise the primal-dual algorithm of Ref. [7] that provides a 2-approximation for both problems. We first show a 2-approximation for the Steiner tree version and then show how to obtain a 2-approximation for the TSP version. Subsequently, we use GW (Goemans-Williamson) to refer to this algorithm.

GW is a *primal-dual* algorithm, that is, the algorithm constructs both a feasible and integral primal and a feasible dual solution for a linear programming formulation of the problem and its dual, respectively. We will consider the version of the problem in which a root vertex r is given and the Steiner tree or the traveling salesman tour will contain r. This is without loss of generality if we can run the algorithm for all possible choices of r.

An integer programming formulation for the Steiner tree problem has a binary variable x_e for all edges $e \in E$: x_e has value 1 if edge e is part of the resulting forest and 0 otherwise. Let us denote by $\mathscr{S}$ all subsets of $V/\{r\}$. The integer programming formulation has a binary variable x_U for each set $U \in \mathscr{S}$. The cost of set U is $\sum_{v \in U} \pi_v$. For a subset $U \in \mathscr{S}$ we define $\delta(U)$ to be the set of all edges that have exactly one endpoint in U. Let T be the set of edges with $x_e = 1$ and let A be the union of sets U for which $x_U = 1$. For any $U \in \mathscr{S}$, any feasible solution must *cross* U at least once, that is, $|\delta(U) \cap T| \geq 1$, or U must be included in the set of vertices that are not spanned, that is, $U \subset A$.

This gives rise to the following integer programming formulation for the Prize-Collecting Steiner Tree problem:

$$\mathtt{opt}_{\mathrm{IP}} = \min \quad \sum_{e \in E} c_e \cdot x_e + \sum_{U \in \mathscr{S}} x_U \cdot \sum_{i \in U} \pi_i \tag{IP}$$

$$\text{s.t.} \quad \sum_{e \in \delta(U)} x_e + \sum_{U' : U \subseteq U'} x_{U'} \geq 1 \quad \forall U \in \mathscr{S} \tag{40.1}$$

$$x_e, x_U \in \{0, 1\} \quad \forall e \in E, \forall U \in \mathscr{S}$$

It is easy to observe that any solution to the Prize-Collecting Steiner Tree problem is an integral solution of this integer linear program: We set $x_e = 1$ for all edges of the Steiner tree T that connects the spanned vertices to the root r. We set $x_U = 1$ for the set U of vertices not spanned by the tree. In the linear programming relaxation of IP we drop the integrality constraints on variables x_e and x_U.

The dual (D) of the linear programming relaxation (LP) of (IP) has a variable y_U for all sets $U \in \mathscr{S}$. There is a constraint for each edge $e \in E$ that limits the total dual assigned to sets $U \in \mathscr{S}$ that contain exactly one endpoint of e to be at most the cost c_e of the edge. There is a constraint for every $U \in \mathscr{S}$ that limits the total dual from subsets of U by at most the total penalty from vertices in U.

$$\mathtt{opt}_{\mathrm{D}} = \max \quad \sum_{U \in \mathscr{S}} y_U \tag{D}$$

$$\text{s.t.} \quad \sum_{U \in \mathscr{S} : e \in \delta(U)} y_U \leq c_e \quad \forall e \in E \tag{40.2}$$

$$\sum_{U' \subseteq U} y_{U'} \leq \sum_{i \in U} \pi_i \quad \forall U \in \mathscr{S} \tag{40.3}$$

$$y_U \geq 0 \quad \forall U \in \mathscr{S}$$

Algorithm GW constructs a primal solution for (IP) and a dual solution for D. The algorithm starts with an infeasible primal solution and reduces the degree of infeasibility as it progresses. At the same time, it creates a dual feasible packing of sets of largest possible total value. The algorithm raises dual variables of certain subsets of vertices. The final dual solution is maximal in the sense that no single set can be raised without violating a constraint of type (40.2) or (40.3).

We can think of an execution of GW as a process over time. Let x^τ and y^τ, respectively, be the primal incidence vector and feasible dual solution at time τ. Initially, $x_e^0 = 0$ for all $e \in E$, $x_U^0 = 0$ for all $U \in \mathscr{S}$, and $y_U^0 = 0$ for all $U \in \mathscr{S}$. In the following we say that an edge $e \in E$ is *tight* if the corresponding constraint (40.2) holds with equality, and that a set U is tight if the corresponding constraint (40.3) holds with equality. We use F^τ to denote the forest formed by the collection of tight edges corresponding to x^τ, and and $\mathscr{S}^\tau$ to denote the collection of tight sets corresponding to x^τ. Assume that the solution x^τ at time τ is infeasible. A set $U \in \mathscr{S}$ is *active* at time τ if it is spanned by a connected component in forest F^τ, there is no tight edge $e \in \delta(U)$, and the corresponding constraint (40.3) is not tight. Let $\mathscr{A}^\tau$ be the collection of sets that are active at time τ. GW raises the dual variables for all sets in $\mathscr{A}^\tau$ uniformly at all times $\tau \geq 0$.

Two kind of events are possible. (i) An edge $e \in \delta(U)$ becomes tight for a set $U \in \mathscr{A}^\tau$. We set $x_e = 1$ and update $\mathscr{A}^\tau$. (Observe that if the tight edge connects U to a component containing r, the newly formed connected component of F^τ is part of $\mathscr{A}^\tau$. If the tight edge connects set U to a component of F^τ that does not contain the root (either active or inactive), the newly formed component is part of $\mathscr{A}^\tau$.) (ii) Constraint (40.3) becomes tight for a set $U \in \mathscr{A}^\tau$. We set $x_U = 1$. GW ends with a reverse pruning phase applied to all sets $U \in \mathscr{S}$ with $x_U = 1$. They are analyzed in order of decreasing time at which the corresponding constraint (40.3) became tight. If the vertices of U are actually connected to the root via tight edges in the current solution, we set $x_U = 0$. The algorithm ends with a tree T connecting a set of vertices to the root r and a set A of vertices for which the penalties are paid. Denote by $c(T, A)$ the total cost of the solution, that is, $c(T, A) = \sum_{e \in T} c_e + \sum_{i \in A} \pi_i$.

Theorem 40.1 **(Goemans and Williamson [7])**

Suppose that algorithm GW *outputs a tree T, a set of vertices A and a feasible dual solution $\{y_U\}_{U \in \mathscr{S}}$. Then*

$$c(T, A) \leq 2 \cdot \sum_{U \in \mathscr{S}} y_U \leq 2 \cdot \texttt{opt}$$

where opt *is the minimum-cost solution for the Prize-Collecting Steiner Tree problem.*

The proof of the above theorem [7] is along the following lines. The total dual of subsets of $U \in \mathscr{S}$ with $x_U = 1$ will pay for the penalties of vertices in A, that is, $\sum_{U: x_{U=1}} \sum_{U' \subseteq U} y_{U'} \leq \sum_{i \in A} \pi_i$, since constraints (40.3) are tight for these sets. Due to the reverse pruning phase, subsets of $U \in \mathscr{S}$ with $x_U = 1$ do not contribute to make tight any edge in T. The cost of T is then paid by twice the total dual of sets $U \in \mathscr{S}$ loading edges of T, that is, those that contribute to making the corresponding constraints (40.2) tight.

GW also provides a 2-approximation for the PTSP when edge costs obey the triangle inequality. First, we run the PCST algorithm with halved penalties. Then, the resulting tree is converted to a tour by doubling every edge and shortcutting the resulting Eulerian tour. For the proof of 2-approximation, we observe that the following integer linear program is a formulation for the problem.

$$\begin{aligned}
\texttt{opt}_{\text{IP}} = \min \ & \sum_{e \in E} c_e \cdot x_e + \sum_{U \in \mathscr{S}} x_U \sum_{i \in U} \frac{\pi_i}{2} && \text{(IP)} \\
\text{s.t.} \ & \sum_{e \in \delta(U)} x_e + \sum_{U' : U \subseteq U'} x_{U'} \geq 2, \quad \forall U \in \mathscr{S} && (40.4) \\
& x_e \in \{0, 1\}, \quad \forall e \in E \\
& x_U \in \{0, 2\}, \quad \forall U \in \mathscr{S}
\end{aligned}$$

Constraints (40.4) impose that each subset must me crossed at least twice unless we pay the penalties for all the vertices of the subset. The dual of the corresponding relaxation is

$$\begin{aligned}
\texttt{opt}_{\text{D}} = \max \ & 2 \cdot \sum_{U \in \mathscr{S}} y_U && \text{(D)} \\
\text{s.t.} \ & \sum_{U \in \mathscr{S} \,:\, e \in \delta(U)} y_U \leq c_e \quad \forall e \in E && (40.5) \\
& \sum_{U' \subseteq U} y_{U'} \leq \sum_{i \in U} \frac{\pi_i}{2} \quad \forall U \in \mathscr{S} && (40.6) \\
& y_U \geq 0 \quad \forall U \in \mathscr{S}
\end{aligned}$$

The cost of the solution provided by `GW` is given by at most twice the cost of tree T and the total penalty of vertices in A. Since twice the total dual collected by the algorithm is a lower bound to the optimal solution, we conclude from the following theorem.

Theorem 40.2 (**Goemans and Williamson** [7])

Suppose that algorithm `GW` *outputs a cycle* C*, a set of vertices* A *and a feasible dual solution* $\{y_U\}_{U\in\mathscr{S}}$*. Then*

$$c(C, A) \le 2 \cdot \left(2 \cdot \sum_{U\in\mathscr{S}} y_U\right) \le 2 \cdot \mathtt{opt}$$

where `opt` *is the minimum-cost solution for the PTSP.*

40.3 The *k*-Minimum Spanning Tree, *k*-TSP and Quota Traveling Salesman Problem

40.3.1 Definitions

k-Minimum Spanning Tree Problem. Given an undirected graph $G = (V, E)$, a tree on G spanning exactly k nodes is called a *k-tree*. Given such a graph with edge costs $c : E \to \mathbb{Q}^+$ and a positive integer k, the (*unrooted*) k-MST problem asks to find a k-tree of minimum total cost. In the *rooted* version of the problem, the k-tree has to include a given root node r.

k-Traveling Salesman Problem. Given an undirected graph $G = (V, E)$, a cycle of G spanning exactly k nodes is called a *k-tour*. Given such a graph with edge costs $c : E \to \mathbb{Q}^+$ satisfying the triangle inequality, a positive integer k and a root node r, the k-TSP asks to find a k-tour including r of minimum total cost.

Quota Traveling Salesman Problem. Given an undirected graph $G = (V, E)$ with vertex weights $w : V \to \mathbb{Z}^+$ and a nonnegative integer Q, a cycle C of G such that $\sum_{v\in C} w(v) \ge Q$ is called a *quota Q-tour*. Given such a graph with edge costs $c : E \to \mathbb{Q}^+$ satisfying the triangle inequality and a root node r, the Quota TSP asks to find a quota Q-tour including r of minimum total cost.

40.3.2 History of the Results

The k-MST problem is known to be an NP-hard problem [14]. Heuristics were given by Cheung and Kumar [15], who studied the problem in the context of communication networks. The first approximation algorithms were considered by Ravi et al. [16], who gave an algorithm achieving an approximation ratio of $O(\sqrt{k})$. Later, this ratio was improved to $O(\log^2 k)$ by Awerbuch et al. [8]. The first constant-ratio algorithm was given by Blum et al. [17]. Subsequently, Garg [5] gave a simple 5-approximation algorithm and a more complicated 3-approximation algorithm, while a 2.5-approximation algorithm for the unrooted case was found by Arya and Ramesh [3]. Arora and Karakostas gave a $(2 + \varepsilon)$-approximation scheme for the rooted version. A 2-approximation by Garg [6] is the current best bound.

We observe that the rooted and the unrooted versions of the k-MST are equivalent with respect to the approximation ratio. In fact, given a c-approximation algorithm for the rooted case it is sufficient to run n times the k-MST algorithm with all possible choices for the root node r, and return the cheapest k-tree found to obtain a c-approximation algorithm for the rooted case . Garg [6] observed that a c-approximation algorithm for the unrooted case gives a c-approximation algorithm for the rooted case.

Some of these works also addressed the k-TSP and Quota TSP. The algorithms for the k-MST by Garg, as well as the scheme by Arora and Karakostas, extend to the k-TSP, thus giving a 2-approximation algorithm for this problem as the current best bound. Finally, as an application of their $O(\log^2 k)$-approximation algorithm for k-MST, Awerbuch et al. give a $O(\log^2(\min(Q, n)))$-approximation algorithm for the Quota TSP, where n is the number of nodes of the graph.

Finally, we remark that a dual version of the TSP is known as the *orienteering problem* [18]. In this problem we are given an edge-weighted graph and a budget and the goal consists in visiting as many vertices of the graph as possible and return to the origin without incurring in a cost greater than the allowed budget. In Ref. [19] a 4-approximation algorithm that makes use of min-cost path algorithms of Ref. [20] is presented for the orienteering problem. In Ref. [21] an improved algorithm leading to a 3-approximation bound for the problem is shown in the context of a more general approach to constrained vehicle routing problem. Other budget versions have been defined also for MST and for TSP [6,19,22].

40.3.3 A 5-Approximation Algorithm for *k*-Minimum Spanning Tree and *k*-Traveling Salesman Problem

In this section, we present and discuss the algorithm by Garg achieving a 5-approximation for the rooted k-MST and its modification yielding the same approximation for the k-TSP. Our analysis follows Chudak et al. [23].

Several assumptions can be made with no loss of generality. First, we can suppose that the edge costs satisfy the triangle inequality, by using well-known techniques [16]. Also, we will assume that the distance from the root to the farthest vertex is a lower bound on the optimum value. It turns out that this is easy to ensure: We can run the algorithm $n-1$ times with all possible choices of a "farthest" vertex, every time disregarding nodes farther than the chosen one, and return the best solution found. The last assumption is that $\texttt{opt} \geq c_0$, where c_0 is the smallest nonzero edge cost. This is not the case only if $\texttt{opt} = 0$, meaning that the optimal solution is a connected component containing r of size k in the graph of zero-cost edges, and the existence of such a component can be easily checked in a preprocessing phase.

A possible formulation of the rooted k-MST as an integer linear problem is the following:

$$
\begin{aligned}
\texttt{opt} = \min \quad & \sum_{e\in E} c_e x_e && \text{(IP)}\\
\text{subject to:} \quad & \sum_{e\in\delta(S)} x_e + \sum_{T:T\supseteq S} z_T \geq 1 \quad \forall S \subseteq V\setminus\{r\} && (1)\\
& \sum_{S:S\subseteq V\setminus\{r\}} |S| z_S \leq n-k && (2)\\
& x_e \in \{0,1\} \quad \forall e \in E\\
& z_S \in \{0,1\} \quad \forall S \subseteq V\setminus\{r\}
\end{aligned}
$$

In the above formulation, r is the root node and $\delta(S)$ the set of edges with exactly one endpoint in S. The variables x_e indicate whether the edge e is included in the tree; the variables z_S indicate whether the set of vertices S is not spanned by the tree. The set of constraints (1) enforces, for each $S \subseteq V\setminus\{r\}$, either some edge of $\delta(S)$ is in the tree or all the vertices in S are not spanned by the tree. Thus, every vertex not in any S such that $z_S = 1$ will be connected to the root r. Constraint (2) enforces at least k vertices to be spanned. Finally, the LP relaxation of this integer program is obtained by replacing the integrality constraints with nonnegativity constraints (in an optimal solution, $x_e \leq 1$ and $z_S \leq 1$ for all e and S).

All the proposed constant approximation algorithms for the k-MST problem use as a subroutine the primal-dual 2-approximation algorithm for the Prize-Collecting Steiner Tree of Goemans and Williamson [7]. This is not by chance, because this problem is essentially the Lagrangean relaxation of the k-MST. Indeed, if we apply Lagrangean relaxation to constraint (2) of the LP relaxation of the k-MST program, we obtain the following:

$$
\begin{aligned}
\min \quad & \sum_{e\in E} c_e x_e + \lambda\left(\sum\nolimits_{S\subseteq V\setminus\{r\}} |S| z_S - (n-k)\right) && \text{(LR)}\\
\text{subject to:} \quad & \sum_{e\in\delta(S)} x_e + \sum_{T:T\supseteq S} z_T \geq 1 \quad \forall S \subseteq V\setminus\{r\}\\
& x_e \geq 0 \quad \forall e \in E\\
& z_S \geq 0 \quad \forall S \subseteq V\setminus\{r\}
\end{aligned}
$$

where $\lambda \geq 0$ is the Lagrangean variable. Apart from the constant term $-\lambda(n-k)$ in the objective function, this is the same as the LP relaxation of the Prize-Collecting Steiner Tree problem with $\pi_v = \lambda$ for all v. Moreover, any solution feasible for the LP relaxation of k-MST is also feasible for (LR), so the value of this program is a lower bound on the cost of an optimal k-MST.

Before discussing Garg's algorithm, we recall that the primal-dual approximation algorithm for the Prize-Collecting Steiner Tree returns a solution (F, A), where F is a tree including the root r, and A is the set of vertices not spanned by F. The algorithm also constructs a feasible solution y for the dual of the LP relaxation of PCST.

Theorem 40.3 **(Goemans and Williamson** [7]**)**

The primal solution (F, A) and the dual solution y produced by the prize-collecting algorithm satisfy

$$\sum_{e\in F} c_e + \left(2 - \frac{1}{n-1}\right)\pi(A) \leq \left(2 - \frac{1}{n-1}\right) \sum_{S\subseteq V\setminus\{r\}} y_S$$

where $\pi(A) = \sum_{v\in A} \pi_v$.

A corollary of Theorem 40.3 is that the prize-collecting algorithm has an approximation ratio of 2, by weak duality and the feasibility of y.

We would like to use the prize-collecting algorithm to solve the k-MST problem. Thus suppose that we run the algorithm with $\pi_v = \lambda$ for all $v \in V$, for some value $\lambda \geq 0$. Then by Theorem 40.3, we obtain (F, A) and y such that

$$\sum_{e\in F} c_e + 2|A|\lambda \leq 2 \sum_{S\subseteq V\setminus\{r\}} y_S \tag{40.7}$$

Consider the dual of the Lagrangean relaxation of the k-MST LP:

$$\begin{aligned} \max \quad & \sum_{S\subseteq V\setminus\{r\}} y_S - (n-k)\lambda & \text{(LR-D)} \\ \text{subject to:} \quad & \sum_{S:e\in\delta(S)} y_S \leq c_e \quad \forall e \in E \\ & \sum_{T:T\subseteq S} y_T \leq |S|\lambda \quad \forall S \subseteq V \setminus \{r\} \\ & y_S \geq 0 \quad \forall S \subseteq V \setminus \{r\} \end{aligned}$$

Since this dual is, apart from the objective function, the same as the dual of the LP relaxation of the PCST instance, the solution y is feasible for this dual, and the value of the objective function is a lower bound on the cost of an optimal k-MST, by weak duality. Subtracting $2(n-k)\lambda$ from both sides of Eq. (40.7)

$$\sum_{e\in F} c_e + 2\lambda\big(|A| - (n-k)\big) \leq 2\left(\sum_{S\subseteq V\setminus\{r\}} y_S - (n-k)\lambda\right) \leq 2\,\texttt{opt}$$

where $\texttt{opt}$ is the cost of an optimal solution to the k-MST instance.

Now if the term $|A| - (n-k)$ is zero, we can conclude that the tree F is a k-tree and has cost no more than twice optimal. Unfortunately, if $|A| - (n-k)$ is positive then the tree F is not feasible, while if it is negative we cannot conclude anything about the approximation ratio. However, it turns out that it is possible to find values of λ such that even if these cases occur, they can be taken care of, although at the cost of resulting in an approximation ratio higher than two.

What Garg's algorithm does is indeed a binary search for these critical values of λ, through a sequence of calls to the prize-collecting algorithm. Notice that if the prize-collecting algorithm is called with $\lambda = 0$, it will return the empty tree spanning only r as a solution, while for $\lambda = \sum_{e\in E} c_e$ it will return a tree spanning all vertices. Thus the initial interval of the binary search will be $[0, \sum_{e\in E} c_e]$, and at every iteration, if the current interval is $[\lambda_1, \lambda_2]$, the prize-collecting algorithm is run with $\lambda = \frac{1}{2}(\lambda_1 + \lambda_2)$. If the returned tree has less than k vertices, we update λ_1 to λ; if it has more than k vertices, we update λ_2 to

λ. Notice that in the lucky event that, at any point, a tree with exactly k vertices is returned, we can stop, since by the above discussion that must be within a factor 2 of optimal. So assume that this event does not happen. We will stop when we have found two values λ_1, λ_2 such that:

(1) $\lambda_2 - \lambda_1 \le \frac{c_0}{2n(n+1)}$ (recall that c_0 is the smallest nonzero edge cost);
(2) for $i = 1, 2$, the prize-collecting algorithm run with λ set to λ_i returns a primal solution (F_i, A_i) spanning k_i vertices and a dual solution $y^{(i)}$, with $k_1 < k < k_2$.

Note that these two values will be found at most after $O(\log \frac{n^2 \sum_e c_e}{c_0})$ calls to the prize-collecting algorithm. The final step of the algorithm is combining the two solutions (F_1, A_1) and (F_2, A_2) into a single k-tree. Solution (F_1, A_1) is within a factor of 2 of optimal, but infeasible, while solution (F_2, A_2) can be easily made feasible but not within a factor of 2 of optimal. More precisely, as a consequence of Theorem 40.3,

$$\sum_{e\in F_1} c_e \le \left(2 - \frac{1}{n}\right)\left(\sum_{S\subseteq V\setminus\{r\}} y_S^{(1)} - |A_1|\lambda_1\right)$$
$$\sum_{e\in F_2} c_e \le \left(2 - \frac{1}{n}\right)\left(\sum_{S\subseteq V\setminus\{r\}} y_S^{(2)} - |A_2|\lambda_2\right)$$

To get a bound on the cost of F_1 and F_2 in terms of $\texttt{opt}$, let

$$\alpha_1 = \frac{n-k-|A_2|}{|A_1|-|A_2|} \text{ and } \alpha_2 = \frac{|A_1|-(n-k)}{|A_1|-|A_2|}.$$

Then $\alpha_1|A_1| + \alpha_2|A_2| = n - k$ and $\alpha_1 + \alpha_2 = 1$, and after defining, for all $S \subseteq V \setminus \{r\}$, $y_S = \alpha_1 y_S^{(1)} + \alpha_2 y_S^{(2)}$, it is possible to prove the following lemma.

Lemma 40.1 (Chudak et al. [23])

$$\alpha_1 \sum_{e\in F_1} c_e + \alpha_2 \sum_{e\in F_2} c_e < 2\texttt{opt}$$

Proof
We omit the proof for brevity; the reader can find it in the overview by Chudak et al. [23]. □

We now show how to obtain a 5-approximation algorithm by choosing one of two solutions. First, if $\alpha_2 \ge \frac{1}{2}$, the tree F_2, besides spanning more than k vertices, satisfies

$$\sum_{e\in F_2} c_e \le 2\alpha_2 \sum_{e\in F_2} c_e \le 4\texttt{opt}$$

by Lemma 40.1. If instead $\alpha_2 < \frac{1}{2}$, the solution is constructed by extending F_1 with nodes from F_2. Let $\ell \ge k_2 - k_1$ be the number of nodes spanned by F_2 but not by F_1. Then we can obtain a path on $k - k_1$ vertices by doubling the tree F_2, shortcutting the corresponding Eulerian tour to a simple tour of the ℓ nodes spanned only by F_2, and choosing the cheapest path of $k - k_1$ vertices from this tour. The resulting path has cost at most

$$2\frac{k-k_1}{k_2-k_1}\sum_{e\in F_2} c_e$$

Notice that this path is disconnected from F_1. However, we can connect it by adding an edge from the root to any node of the set, costing at most $\texttt{opt}$ by one of the assumptions at the beginning of the section. Since

$$\frac{k-k_1}{k_2-k_1} = \frac{n-k_1-(n-k)}{n-k_1-(n-k_2)} = \frac{|A_1|-(n-k)}{|A_1|-|A_2|} = \alpha_2$$

the total cost of the produced solution is bounded by

$$\sum_{e\in F_1} c_e + 2\alpha_2 \sum_{e\in F_2} c_e + \mathtt{opt} \le 2\left(\alpha_1 \sum_{e\in F_1} c_e + \alpha_2 \sum_{e\in F_2} c_e\right) + \mathtt{opt} \le 4\mathtt{opt} + \mathtt{opt}$$

using Lemma 40.1 and the fact that $\alpha_2 < \frac{1}{2}$ implies $\alpha_1 > \frac{1}{2}$.

As for the k-TSP, it suffices to run the Prize-Collecting subroutine with halved penalties and then shortcut the Eulerian walk obtained after doubling the k-tree found, in the same way as we went from the Prize-Collecting Steiner Tree to the PTSP.

40.3.4 From the *k*-MST to the Quota Traveling Salesman Problem

In this section we will describe a 5-approximation algorithm for the Quota TSP. However, the discussion will be easier if we consider the following problem first.

Quota Minimum Spanning Tree Problem
Given an undirected graph $G = (V, E)$ with vertex weights $w : V \rightarrow \mathbb{Z}^+$ and a positive integer Q, a tree F of G such that $\sum_{v\in F} w(v) \ge Q$ is called a *quota Q-tree.* Given such a graph with edge costs $c : E \rightarrow \mathbb{Q}^+$ and a root node r, the Quota MST Problem asks to find a quota Q-tree including r of minimum total cost.

Theorem 40.4 (**Ausiello et al.** [9])

There is a 5-approximation algorithm for the Quota MST Problem.

The idea behind the theorem is that we can run the 5-approximation algorithm for the k-MST by Garg, but instead of setting uniformly the penalties to λ, we set $\pi_v = \lambda \cdot w_v$ when calling the Prize-Collecting Steiner Tree subroutine. The two solutions obtained at the end of the binary search phase can then be patched essentially as before.

Now, we can obtain an algorithm for the Quota TSP in the same way as we went from the Prize-Collecting Steiner Tree to the PTSP in Section 40.2. That is, it is sufficient to run the Prize-Collecting subroutine with $\pi_v = \frac{1}{2}\lambda w_v$. The analysis remains the same.

40.4 The Prize-Collecting Traveling Salesman Problem

40.4.1 Definitions

Prize-Collecting Traveling Salesman Problem
Given an undirected graph $G = (V, E)$ with vertex weights $w : V \rightarrow \mathbb{Z}^+$, vertex penalties $\pi : V \rightarrow \mathbb{Q}^+$, and a nonnegative integer Q, a cycle C of G such that $\sum_{v\in C} w(v) \ge Q$ is called a *quota Q-tour.* Given such a graph with edge costs $c : E \rightarrow \mathbb{Q}^+$ satisfying the triangle inequality and a root node r, the PCTSP asks to find a quota Q-tour $T = (V_T, E_T)$ including r that minimizes

$$c(T) = \sum_{e\in E_T} c_e + \sum_{v\in V\setminus V_T} \pi_v$$

40.4.2 History of the Results

In the general form given here, the PCTSP was first formulated by Balas [1,24], who gave structural properties of the PCTS polytope as well as heuristics. The problem arose during the task of developing daily schedules for a steel rolling mill.

The only results on guaranteed heuristics for the PCTSP are due to Awerbuch et al. [8]. They give polynomial-time algorithm with an approximation ratio of $O(\log^2(\min(Q, n)))$, where n is the number of vertices of the graph and Q is the required vertex weight to be visited. However, the PCTSP contains as special cases both the PTSP and the k-TSP, which received more attention in the literature (for the history of results on these problems, the reader can refer to the previous sections). From some recent results on these problems, we derive a constant-approximation algorithm for the general PCTSP in the following section.

40.4.3 A Constant-Factor Approximation Algorithm for PCTSP

A simple idea exploited by the algorithm of Awerbuch et al. [8] is that, for a given instance I of the PCTSP, the following quantities constitute lower bounds on the cost opt of an optimal solution:

(1) the cost optp of an optimal solution to a PTSP instance I_p defined on the same graph and having the same penalties as in the PCTSP instance (since a feasible solution to I is also feasible for I_p and has the same cost);
(2) the cost optq of an optimal solution to a Quota TSP instance I_q defined on the same graph and having the same weights and quota as in I (since every feasible solution to I can be turned into a feasible solution for I_q of at most the same cost).

Thus, to approximate an optimal solution to the PCTSP instance I we can:

(1) run an α-approximation algorithm for PTSP on I_p to obtain a tour T_p such that $c(T_p) \leq \alpha \cdot \texttt{optp}$;
(2) run a β-approximation algorithm for Quota TSP on I_q to obtain a tour T_q such that $c(T_q) \leq \beta \cdot \texttt{optq}$;
(3) concatenate T_p and T_q to obtain a tour T feasible for the PCTSP instance I of cost

$$c(T) \leq c(T_p) + c(T_q) \leq \alpha \cdot \texttt{optp} + \beta \cdot \texttt{optq} \leq (\alpha + \beta)\texttt{opt}$$

This means that, by using the best algorithms currently known for PTSP and Quota TSP, we can obtain a constant-factor approximation to the PCTSP.

40.5 The Minimum Latency Problem and Graph Searching

Suppose that a plumber receives calls from various customers and decides to organize a tour of the customers for the subsequent day; for sake of simplicity let us also assume that, at each visit, the time the plumber needs to fix the customer's problem is constant. A selfish plumber would decide to schedule his tour in such way as to minimize the overall time he takes to serve all customers and come back home; such approach would require the solution of an instance of TSP. Alternatively, a nonselfish plumber would decide to schedule his tour in such a way to minimize the average time customers have to wait for his visit the day after. In this case he will have to solve an instance of the so-called *traveling repairman problem* (TRP).

The TRP problem is more frequently known in the literature as *Minimum Latency Problem* (MLP) [25], but it is also known as *school-bus driver problem* [26] and the *delivery man problem* [27,28]. Strictly related to MLP is the so-called *graph searching problem* (GSP) [29]. In such problem we assume that a single prize is hidden in a vertex of an edge-weighted graph and the vertices are labeled with the probability that the prize is stored in the vertex. The goal is to minimize the expected cost to find the prize by exploring all vertices of the graph. The relationship between MLP and GSP is discussed in Ref. [9].

40.5.1 Definitions

Minimum Latency Problem

Given an undirected graph $G = (V, E)$, with edge costs $c : E \to \mathbb{Q}^+$ satisfying the triangle inequality, let T be a tour that visits the vertices in some order. The *latency* $l_{v_i,T}$ of a vertex $v_i \in T$ is the cost of the prefix of T ending in v_i. The MLP asks to find a tour T such that the sum of the latencies of all vertices along T is minimum.

40.5.2 History of the Results

The problem has been shown to be NP-hard by Sahni and Gonzalez [30]. In Ref. [31] Afrati et al. showed that the problem can be solved in polynomial time on trees with bounded number of leaves. Recently, Sitters [32] has shown that the problem is NP-hard also in the case of general weighted trees.

From the approximability point of view, the MLP is, clearly, as the TSP, hard to approximate for any given constant ratio on general graphs [30] (i.e., when the triangle inequality does not hold), while in the case of metric spaces it can be shown to be APX-complete, that is, it allows approximation algorithms but does not allow approximation schemes.

The first constant-factor approximation algorithm for the MLP on general metric spaces has been presented by Blum et al. [25] who show that given a c-approximate algorithm for the k-MST then there exists a $8c$-approximation ratio for the MLP. Subsequently, Goemans and Kleinberg [33] showed that the constant 8 above can be lowered to 3.59, thus implying a 7.18-approximation algorithm for MLP. The best current bound is 3.59 is given by Chaudhuri et al. in Ref. [20].

Arora and Karakostas [34] showed the existence of a quasi-polynomial-time approximation algorithm when the input graph is a tree; to compute a $(1+\varepsilon)$-approximation the algorithm requires time $n^{O(\log n/\varepsilon)}$ time.

We finally remark that the problem has also been extended to the case of k repairmen. Namely, Fakcharoenphol et al. [35] showed the first constant approximation algorithm for the problem. This result has been improved to 8.49-approximation by Chaudhuri et al. in Ref. [20].

40.5.3 A 3.59-Approximation Algorithm for the Minimum Latency Problem

We first present the algorithm proposed by Goemans and Kleinberg in Ref. [33] that gives a 7.18-approximation algorithm. The procedure proposed by the authors computes, for every $j = 1, 2, \ldots, n$ the tour T_j of minimum length that visits j vertices. Then we have to concatenate a subsequence of the tours to form the desired tour. Clearly, the goal is to select those values $j_1, \ldots, j_m$ such that the latency of the final tour obtained by stitching together tours $T_{j_1} \ldots T_{j_m}$ is minimized.

Let d_{j_i} and p_i be the length of tour T_{j_i} and the number of new vertices visited during the same tour, respectively. It is simple to show that the following claim holds:

$$\sum_{i=1}^{m} p_i d_{j_i} \leq \sum_{i=1}^{m} (j_i - j_{i-1}) d_{j_i}$$

Note that if, for every i, the tours T_{j_i} and $T_{j_{i-1}}$ were nested, the above inequality would be trivially satisfied, but a careful analysis of the contributions involved on the right and the left-hand sides of the inequality may convince the reader that such inequality is also true when the tours are not nested. It follows that for a number of vertices equal to $\sum_{k=1}^{i} p_k - j_i$ we sum a contribution at most d_{j_k} on the left-hand side of the equation while a contribution larger than d_{j_k} on the right-hand side of the equation. Moreover, each tour T_{j_i} is traversed in the direction that minimizes the total latency of the vertices discovered during tour T_{j_i}. This allows to rewrite the total latency of the tour obtained by concatenating $T_{j_1}, \ldots, T_{j_m}$ as

$$\sum_i \left(n - \sum_{k=1}^{i} p_k \right) d_{j_i} + \frac{1}{2} \sum_i p_i d_{j_i}$$

$$\leq \sum_i (n - j_i) d_{j_i} + \frac{1}{2} \sum_i (j_i - j_{i-1}) d_{j_i}$$

$$= \sum_i \left(n - \frac{j_{i-1} + j_i}{2} \right) d_{j_i}$$

The formula above allows to rewrite the total latency of the algorithm only in terms of the indices j_i and of the length d_{j_i}, independently from the number of new vertices discovered during each tour. A complete digraph of $n+1$ vertices is then constructed in the following way. Arc (i, j) goes from $min(i, j)$ to $max(i, j)$ and has length $(n - \frac{i+j}{2})d_{j_i}$. The algorithm computes a shortest path from node 0 to node n. Assume that the path goes through nodes $0 = j_0 < j_1 < \cdots < j_m = n$. The tour is then obtained by concatenating tours $T_{j_1}, \ldots, T_{j_m}$.

The obtained solution is compared against the following lower bound $OPT \geq \sum_{k=1}^{n} \frac{d_k}{2}$. This lower bound follows from the observation that the kth vertex cannot be visited before $d_k/2$ in any optimum tour. The approximation ratio of the algorithm is determined by bounding the maximum over all the possible set of distances $d_1, \ldots, d_n$ of the ratio between the shortest path in G_n and the lower bound on the optimum solution. This value results to be smaller than 3.59.

Theorem 40.5 (**Goemans and Kleinberg** [33])

Given a c-approximation algorithm for the problem of finding an a tour of minimum length spanning at least k vertices on a specific metric space, then there exists a 3.59c-approximation algorithm for the MLP *on the same metric space.*

Again by making use of the 2-approximation algorithm of [6] for k-MST and k-TSP we may achieve a ratio 7.18 for MLP.

With respect to the results we have seen so far, a remarkable step forward has been achieved by Chaudhuri et al. in Ref. [20]. Using techniques from Garg [5], Arora and Karakostas [2], and Archer et al. [36], the authors are able to find a k-MST whose cost is no more than $(1+\varepsilon)$ the cost of the minimum path visiting k vertices. Since such a cost is a lower bound on the latency of a k tour the result implies the following theorem.

Theorem 40.6 (**Chaudhuri et al.** [20])

There exists a 3.59-*approximation algorithm for the* MLP *on general metric spaces.*

By the arguments provided in Ref. [9] the same approximation bound also holds for GSP.

References

[1] Balas, E., The prize collecting traveling salesman problem, *Networks*, 19, 621, 1989.

[2] Arora, S. and Karakostas, G., A $2+\varepsilon$ approximation algorithm for the k-MST problem, *Proc. SODA*, 2000, p. 754.

[3] Arya, S. and Ramesh, H., A 2.5-factor approximation algorithm for the k-MST problem, *Inf. Proc. Lett.*, 65(3), 117, 1998.

[4] Blum, A., Ravi, R., and Vempala, S., A constant-factor approximation algorithm for the k-MST problem, *Proc. of STOC*, 1996, p. 442.

[5] Garg, N., A 3-approximation for the minimum tree spanning k vertices, *Proc. FOCS*, 1996, p. 302.

[6] Garg, N., Saving an epsilon: A 2-approximation for the k-MST problem in graphs, *Proc. STOC*, 2005, p. 396.

[7] Goemans, M. and Williamson, D. P., A general approximation technique for constrained forest problems, *SIAM J. Comput.*, 24(2), 296, 1995.

[8] Awerbuch, B., Azar, Y., Blum, A., and Vempala, S., New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen, *SIAM J. Comput.*, 28(1), 254, 1999.

[9] Ausiello, G., Leonardi, S., and Marchetti-Spaccamela, A., On salesmen, repairmen, spiders, and other traveling agents, *Proc. 4th Italian Conf. on Algorithms and Complexity*, Lecture Notes in Computer Science, Vol. 1767, Springer, Berlin, 2000, p. 1.

[10] Gabow, H. N. and Pettie, S., The dynamic vertex minimum problem and its application to clustering-type approximation algorithms, *Proc. 8th Scandinavian Workshop on Algorithm Theory*, Lecture Notes in Computer Science, Vol. 2368, Springer, Berlin, 2002, p. 190.

[11] Bienstock, D., Goemans, M. X., Simchi-Levi, D., and Williamson, D., A note on the prize collecting traveling salesman problem, *Math. Prog.*, 59(3), 413, 1993.

[12] Bern, M. W. and Plassmann, P. E., The Steiner problem with edge lengths 1 and 2, *Inf. Proc. Lett.*, 32(4), 171, 1989.

[13] Papadimitriou, C. H. and Yannakakis, M., The traveling salesman problem with distances one and two, *Math. Oper. Res.*, 18(1), 1, 1993.

[14] Fischetti, M., Hamacher, H. W., Jørnsten, K., and Maffioli, F., Weighted *k*-cardinality trees: complexity and polyhedral structure, *Networks*, 24(1), 11, 1994.

[15] Cheung, S. Y. and Kumar, A., Efficient quorumcast routing algorithms, *Proc. of INFOCOM*, 2, 1994, 840.

[16] Ravi, R., Sundaram, R., Marathe, M. V., Rosenkrantz, D. J., and Ravi, S. S., Spanning trees short or small, *SIAM J. Disc. Math.*, 9(2), 178, 1996.

[17] Blum, A., Ravi, R., and Vempala, S., A constant-factor approximation algorithm for the *k*-MST problem, *JCSS*, 58(1), 101, 1999.

[18] Golden, B. L., Levy, L., and Vohra, R., The orienteering problem, *Nav. Res. Logistics*, 34, 307, 1987.

[19] Blum, A., Chawla, S., Karger, D. R., Lane, T., Meyerson, A., and Minkoff, M., Approximation algorithms for orienteering and discounted-reward TSP, *Proc. FOCS*, 2003, p. 46.

[20] Chaudhuri, K., Godfrey, B., Rao, S., and Talwar, K., Paths, trees, and minimum latency tours, *Proc. FOCS*, 2003, p. 36.

[21] Bansal, N., Blum, A., Chalasani, P., and Meyerson, A., Approximation algorithms for deadline-tsp and vehicle routing with time-windows, *Proc. STOC*, 2004, p. 166.

[22] Johnson, D. S., Minkoff, M., and Phillips, S., The prize collecting steiner tree problem: theory and practice, *Proc. SODA*, 2000, p. 760.

[23] Chudak, F. A., Roughgarden, T., and Williamson, D. P., Approximate *k*-MSTs and *k*-Steiner trees via the primal-dual method and Lagrangean relaxation, *Math. Prog.*, 100(2), 411, 2004.

[24] Balas, E., The prize collecting traveling salesman problem: II. Polyhedral results, *Networks*, 25, 199, 1995.

[25] Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, W. R., Raghavan, P., and Sudan, M., The minimum latency problem, *Proc. of STOC*, 1994, p. 163.

[26] Will, T. G., Extremal Results and Algorithms for Degree Sequences of Graphs, Ph.D. thesis, University of Illinois at Urbana-Champaign, 1993.

[27] Minieka, E., The deliver man problem on a tree network, *Ann. Oper. Res.*, 18, 261, 1989.

[28] Fischetti, M., Laporte, G., and Martello, S., The delivery man problem and cumulative matroids, *Oper. Res.*, 41(6), 1055, 1993.

[29] Koutsoupias, E., Papadimitriou, C. H., and Yannakakis, M., Searching a fixed graph, *Proc. of ICALP*, Lecture Notes in Computer Science, Vol. 1099, Springer, 1996, p. 280.

[30] Sahni, S. and Gonzalez, T. F., P-complete approximation problems, *JACM*, 23(3), 555, 1976.

[31] Afrati, F. N., Cosmadakis, S. S., Papadimitriou, C. H., Papageorgiou, G., and Papakostantinou, N., The complexity of the travelling repairman problem, *Informatique Théorique Appl.*, 20(1), 79, 1986.

[32] Sitters, R., The minimum latency problem is NP-hard for weighted trees, *Proc. 9th Integer Programming and Combinatorial Optimization Conf.*, 2002, p. 230.

[33] Goemans, M. and Kleinberg, J., An improved approximation ratio for the minimum latency problem, *Math. Prog.*, 82(1), 111, 1998.

[34] Arora, S. and Karakostas, G., Approximation schemes for minimum latency problems, *SIAM J. Comput.*, 32(5), 1317, 2003.

[35] Fakcharoenphol, J., Harrelson, C., and Rao, S., The *k*-traveling repairman problem, *Proc. SODA*, 2003, p. 655.

[36] Archer, A., Levin, A., and Williamson, D. P., A Faster, Better Approximation Algorithm for the Minimum Latency Problem, Technical report 1362, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 2003.

41

A Development and Deployment Framework for Distributed Branch and Bound

Peter Cappello
University of California, Santa Barbara

Christopher James Coakley
University of California, Santa Barbara

41.1 Introduction

Branch-and-bound intelligently searches the set of feasible solutions to a combinatorial optimization problem: It, in effect, proves that the optimal solution is found without necessarily examining all feasible solutions. The feasible solutions are not given. They can be generated from the problem description. However, doing so usually is computationally infeasible: The number of feasible solutions typically grows exponentially as a function of the size of the problem input. For example, the set of feasible tours in a symmetric Traveling Salesman Problem (TSP) of a complete graph with 23 nodes is 22!/2 or around 8×10^{14} tours. The space of feasible solutions is progressively partitioned (branching), forming a search *tree*. Each tree node has a partial feasible solution. The node *represents* the set of feasible solutions that are extensions of its partial solution. For example, in a TSP branch and bound, a search tree node has a partial tour, representing the set of all tours that contain that partial tour. As branching continues (progresses down the problem tree), each search node has a more complete partial solution, and thus represents a smaller set of feasible solutions. For example, in a TSP branch and bound, a tree node's children each represents an extension of the partial tour to a more complete tour (e.g., one additional city or one additional edge). As one progresses down the search tree, each node represents a larger partial tour. As the size of a partial tour increases, the number of full tours containing the partial tour clearly decreases.

In traversing the search tree, we may come to a node that represents a set of feasible solutions, all of which are provably more costly than a feasible solution already found. When this occurs, we *prune* this node of the search tree: We discontinue further exploration of this set of feasible solutions. In the example

```
activeSet = { originalTask };
u = infinity; // u = the cost of the best solution known
currentBest = null;
while ( ! activeSet.isEmpty() ) {
    k = remove some element of the activeSet;
    children = generate k's children;
    for each element of children {
        if ( element's lower bound <= u )
            if ( element is a complete solution ) {
                u = element's cost;
                currentBest = element;
            }
            else
                add element to activeSet;
    }
}
```

FIGURE 41.1 A sequential algorithm for branch and bound.

of the TSP, the cost of any feasible tour that has a given partial tour surely can be bounded from below by the cost of the partial tour: the sum of the edge weights for the edges in the partial tour. (In our experiments, we use a Held–Karp lower bound, which is stronger but more computationally complex.) If the lower bound for a node is *higher* than the current upper bound (i.e., best known complete solution), then the cost of all complete solutions (e.g., tours) represented by the node is higher than a complete solution that is already known: The node is pruned (see Papadimitriou and Steiglitz [1] for a more complete discussion of branch-and-bound). Figure 41.1 gives a basic, sequential branch-and-bound algorithm.

Branch-and-bound may be easily modified to generate suboptimal solutions. The total search time decreases as the desired accuracy decreases.

The framework that we present here is designed for deployment in a distributed setting. Moreover, the framework supports *adaptive parallelism*: During the execution, the set of compute servers can grow (if new compute servers become available) or shrink (if compute servers become unavailable or fail): the branch-and-bound computation thus cannot assume a fixed number of compute servers.

The branch-and-bound computation is decomposed into tasks, each of which is executed on a compute server: Each element of the active set (see Figure 41.1) is a task that, in principle, can be scheduled for execution on any compute server. Indeed, parallel efficiency requires load balancing of tasks among compute servers. This distributed setting implies the following compute server requirements:

- Tasks (activeset elements) are generated during the computation—they cannot be scheduled a priori.
- When a compute server discovers a new best cost, it must be propagated to the other compute servers.
- Detecting termination requires “knowing” when all branches (children) have been either fully examined or pruned. In a distributed setting, the implied communication must not be a bottleneck.

Our goal is to facilitate the development of branch-and-bound computations for deployment as a distributed computation. We provide a development–deployment infrastructure that requires the developer to write code for *only* the particular aspects of the branch-and-bound computation under development, primarily the branching rule, the lower bound computation, and the upper bound computation. We present this framework and some experimental results of its application to a medium complexity TSP code running on a beowulf cluster.

41.2 Related Work

Held et al. give a short history of the TSP [2]. In it, they note that, in 1963, Little et al. [3] were the first to use the term “branch and bound” to describe their enumerative procedure for solving TSP instances. As we understand it, Little et al. [3] and Land and Doig [4] independently discovered the technique of branch and bound. This discovery led to "a decade of enumeration."

Parallel branch and bound also has been widely studied. See, for example, Refs. [5,6]. Rather early on it was discovered that there are speedup anomalies in parallel branch and bound [7]: Completion times are not monotonically nonincreasing as a function of the number of processors. In the discussion that follows, let T denote the search tree, c^* the cost of a minimum-cost leaf in T, $T^* \subseteq T$ the subtree of T whose nodes cost less than or equal to c^*, n the number of nodes in T^*, and h the height of T^*. Karp and Zhang [8] present a universal randomized method called Local Best-First Search for parallelizing sequential branch-and-bound algorithms. When executing on a completely connected, message-passing multiprocessor, the method's computational complexity is asymptotically optimal with high probability, $O(n/p + h)$, where p is the number of processors. The computational complexity of maintaining the local data structure and the communication overhead is ignored in their analysis. When $n > p^2 \log p$, Liu et al. [9] give a method for branch and bound that is asymptotically optimal with high probability, assuming that inter-processor communication is controlled by a central First In, First Out (FIFO) arbiter. Herley et al. [10] give a deterministic parallel algorithm for branch and bound based on the parallel heap selection algorithm of Frederickson [11], combined with a parallel priority queue. The complexity of their method is $O(n/p + h \log^2(np))$ on an EREW-PRAM, which is optimal for $h = O(n/(p \log^2(np)))$. This bound includes communication costs on an EREW-PRAM.

Distributed branch and bound has also been widely studied. Tschöke et al. [12] contributed experimental work on distributed branch and bound for TSP using over 1000 processors. When the number of processors gets large, fault tolerance becomes an issue. Yahfoufi and Dowaji [13] present perhaps the first distributed fault-tolerant branch-and-bound algorithm.

There also has been a lot of work on what might be called programming frameworks for distributed branch-and-bound computation. This occurs for two reasons: (1) branch and bound is best seen as a metaalgorithm for solving large combinatorial optimization problems: It is a framework that must be completed with problem-specific code and (2) programming a fault-tolerant distributed system is sufficiently complex to motivate a specialization of labor: distributed systems research versus operations research. In 1995, Shinano et al. [14] presented a Parallel Utility for Branch and Bound (PUBB) based on the C programming language. They illustrate the use of their utility on TSP and 0/1 ILP. They introduce the notion of a Logical Computing Unit (LCU). Although in parts of their paper, an LCU sounds like a computational task, we are persuaded that it most closely resembles a processor, based on their explanation of its use: "The master process maintains in a queue, all the subproblems that are likely to lead to an optimal solution. As long as this queue is not empty and an idle LCU exists, the master process selects subproblems and assigns them to an idle LCU for evaluation one after the other." When discussing their experimental results, they note: "The results indicate that, up to using about 10 LCUs, the execution time rapidly decreases as more LCUs are added. When the number of LCUs exceeds about 20, the computing time for one run, remains almost constant." Indeed, from their Figure 9 (in Ref. [14]) , we can see that PUBB's parallel efficiency steadily goes down when the number of LCUs is above 10, and is well below 0.5, when the number of LCUs is 55. Aversa et al. [15] report on the Magda project for mobile agent programming with parallel skeletons. Their divide-and-conquer skeleton is used to implement branch and bound, which they provide experimental data for up to eight processors. Moe [16] reports on GRIBB, and infrastructure for branch and bound on the Internet. Experimental results on an SGI Origin 2000 with 32 processors machines show good speedups when the initial bound is tight, and $\sim$ 67% of ideal speedup, when a simple initial bound is used. Dorta et al. [17] present C++ skeletons for divide-and-conquer and branch-and-bound, where deployment is intended for clusters. Their experiments, using a 0/1 knapsack problem of size 1000, on a Linux cluster with seven processors, the average speedup was 2.25. On an Origin 3000 with 16 processors, the average speedup was 4.6. On a Cray T3E with 128 processors, the average speedup was 5.02. They explain "Due to the fine grain of the 0/1 knapsack problem, there is no lineal increase in the speed up when the number of processor increase. For large numbers of processors the speed up is poor."

Neary et al. [18] and Neary and Cappello [19] present an infrastructure/framework for distributed computing, including branch and bound, that tolerates faulty compute servers, and is in pure Java, allowing application codes to run on a heterogeneous set of machine types and operating systems. They experimentally achieved 50% of ideal speedup for their TSP code, when running on 1000 processors. Their schemes for termination detection and fault tolerance of a branch-and-bound computation both exploit its *tree-structured* search space. The management of these schemes is centralized. Iamnitchi and Foster [20] build on this idea of exploiting branch and bound's tree-structured search space, producing a branch-and-bound-specific fault tolerance scheme that is distributed, although they provide only simulation results.

41.3 The Deployment Architecture

JICOS, a Java-centric network computing service that supports high-performance parallel computing, is an ongoing project that virtualizes compute cycles, stores/coordinates *partial* results—supporting fault tolerance, is partially self-organizing, may use an open grid services architecture [21,22] front end for service discovery (not presented), is largely independent of hardware/OS, and is intended to scale from a LAN to the Internet. JICOS is designed to **support scalable, adaptively parallel computation** (i.e., the computation's organization reduces *completion* time, using many *transient* compute servers, called *hosts*, that may join and leave during a computation's execution, with high system efficiency, regardless of how many hosts join/leave the computation); **tolerate basic faults**: JICOS must tolerate host failure and network failure between hosts and other system components; **hide communication latencies**, which may be long, by overlapping communication with computation. JICOS comprises three service component classes.

Hosting Service Provider. JICOS clients (i.e., processes seeking computation done on their behalf) interact solely with the hosting service provider (HSP) component. A client logs in, submits computational tasks, requests results, and logs out. When interacting with a client, the HSP thus acts as an agent for the entire network of service components. It also manages the network of task servers described below. For example, when a task server wants to join the distributed service, it first contacts the HSP. The HSP tells the task server where it fits in the task server network.

Task Server. This component is a store of task objects. Each task object, which has been spawned but has not yet been computed, has a representation on some task server. Task servers balance the load of ready tasks among themselves. Each task server has a number of hosts associated with it. When a host requests a task, the task server returns a task that is ready for execution, if any are available. If a host fails, the task server reassigns the host's tasks to other hosts.

Host. A host (aka compute server) joins a particular task server. Once joined, each host repeatedly requests a task for execution, executes the task, returns the results, and requests another task. It is the central service component for virtualizing compute cycles.

When a client logs in, the HSP propagates that log-in to all task servers, who in turn propagate it to all their hosts. When a client logs out, the HSP propagates that log-out to all task servers, which *aggregate* resource consumption information (execution statistics) for each of their hosts. This information, in turn, is aggregated by the HSP for each task server, and returned to the client. Currently, the task server network topology is a torous. However, scatter/gather operations, such as log-in and log-out, are transmitted via a task server *tree*: a subgraph of the torous (see Figure 41.2).

Task objects *encapsulate* computation: Their inputs and outputs are managed by JICOS. Task execution is idempotent, supporting the requirement for host transience and failure recovery. Communication latencies between task servers and hosts are reduced or hidden via task caching, task prefetching, and task execution on task servers for selected task classes.

41.3.1 Tolerating Faulty Hosts

To support self-healing, all proxy objects are leased [23,24]. When a task server's lease manager detects an expired host lease and the offer of renewal fails, the host proxy: (1) returns the host's tasks for reassignment and (2) is deleted from the task server. Because of explicit continuation passing, recomputation is

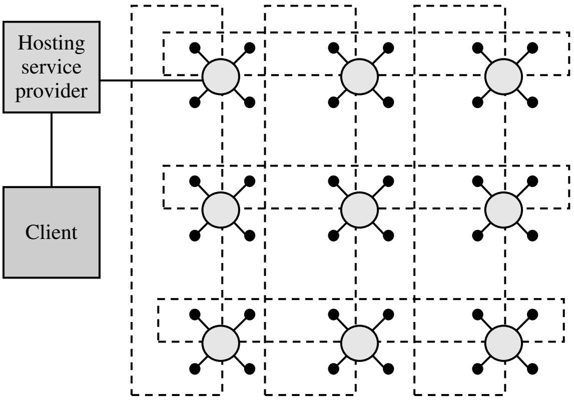


FIGURE 41.2 A JICOS system that has nine task servers. The task server topology, a 2D torous, is indicated by the broken lines. In the figure, each task server has four associated hosts (the little black discs). An actual task server might serve about 40 hosts (although our experiments indicate that 128 hosts/task server is not too much). The client interacts *only* with the HSP.

minimized: Systems that support divide-and-conquer but do not use explicit continuation passing [25], such as Satin [26], need to recompute some task decomposition computations, even if they completed successfully. In some applications, such as sophisticated TSP codes, decomposition can be computationally complex. On Jicos, only the task that was currently being executed needs to be recomputed. This is a substantial improvement. In the TSP instance that we use for our performance experiments, the average task time is 2 s. Thus, the recomputation time for a failed host is, in this instance, a mere 1 s, on average.

41.4 Performance Considerations

JICOS's API includes a simple set of *application-controlled* directives for improving performance by reducing communication latency or overlapping it with task execution.

Task caching. When a task constructs subtasks, the first constructed subtask is cached on its host, obviating its host's need to ask the TaskServer for its next task. *The application programmer thus implicitly controls which subtask is cached.*

Task prefetching. The *application* can help hide communication latency via task prefetching:

> *Implicit.* A task that never constructs subtasks is called *atomic.* The Task class has a Boolean method, *isAtomic.* The default implementation of this method returns true, if and only if the task's class implements the marking interface, *Atomic.* Before invoking a task's *execute* method, a host invokes the task's *isAtomic* method. If it returns true, the host prefetches another task via another thread before invoking the task's execute method.
>
> *Explicit.* When a task object whose *isAtomic* method returned false (it did not know prior to the invocation of its *execute* method that it would not generate subtasks) nonetheless comes to a point in its *execute* method when it knows that it is not going to construct any subtasks, it can invoke its environment's *prefetch* method. This causes its host to request a task from the task server in a separate thread.

Task prefetching overlaps the host's execution of the current task with its request for the next task. Application-directed prefetching, both implicit and explicit, thus motivates the programmer to (1) identify atomic task classes and (2) constitute atomic tasks with compute time that is at least as long as a Host–TaskServer round trip (on the order of 10s of milliseconds, depending on the size of the returned task, which affects the time to marshal, send, and unmarshal it).

Task server computation. When a task's encapsulated computation is little more complex than reading its inputs, it is faster for the task server to execute the task itself than to send it to a host for execution. This is because the time to marshal and unmarshal the input plus the time to marshal and unmarshal the result is less than the time to simply compute the result (not to mention network latency). Binary Boolean operators, such as min, max, sum (typical linear-time gather operations), should execute on the task server. All Task classes have a Boolean method, executeOnServer. The default implementation returns true, if and only if the task's class implements the marking interface, *ExecuteOnServer*. When a task is ready for execution, the task server invokes its executeOnServer method. If it returns true, the task server executes the task itself: *The application programmer controls the use of this important performance feature.*

Taken together, these features reduce or hide much of the delay associated with Host–TaskServer communication.

41.5 The Computational Model

Computation is modeled by a *directed acyclic graph* (DAG) whose nodes represent tasks. An arc from node v to node u represents that the output of the task represented by node v is an input to the task represented by node u. A computation's tasks all have access to an *environment* consisting of an immutable *input* object and a mutable *shared* object. The semantics of "shared" reflects the envisioned computing context—a computer network: The object is shared *asynchronously*. This limited form of sharing is of value in only a limited number of settings. However, branch and bound is one such setting, constituting a versatile paradigm for coping with computationally intractable optimization problems.

41.6 The Branch-and-Bound API

Tasks correspond to nodes in the search tree: Each task gives rise to a set of smaller subtasks, until it represents a node in the search tree that is small enough to be explored by a single compute server. We refer to such a task as *atomic*; it does not decompose into subtasks.

41.6.1 The Environment

For branch-and-bound computations, the environment *input* is set to the problem instance. For example, in a TSP, the input can be set to the distance matrix. Doing so materially reduces the amount of information needed to describe a task, which reduces the time spent to marshal and unmarshal such objects.

The cost of the least cost known solution at any point in time is shared among the tasks: It is encapsulated as the branch-and-bound computation's *shared* object (see IntUpperBound below). In branch and bound, this value is used to decide if a particular subtree of the search tree can be pruned. Thus, sharing the cost of the least cost known solution enhances the pruning ability of concurrently executing tasks that are exploring disjoint parts of the search tree. Indeed, this improvement in pruning is essential to the efficiency of parallel branch-and-bound. When a branch-and-bound task finds a complete solution whose cost is less than the current least cost solution, it sets the shared object to this new value, which implicitly causes JICOS to propagate the new least cost throughout the distributed system.

41.6.2 The JICOS Branch-and-Bound Framework

The classes comprising the JICOS branch-and-bound framework are based on two assumptions:

- The branch-and-bound problem is formulated as a minimization problem. Maximization problems can be typically reformulated as minimization problems.
- The cost can be represented as in int.

Should these two assumptions prove troublesome, we will generalize this framework.

Before giving the framework, we describe the problem-specific class that the *application developer must provide*: A class that implements the *Solution* interface. This class represents nodes in the search tree: A Solution object is a partial feasible solution. For example, in a TSP, it could represent a partial tour. Since it represents a node in the search tree, its children represent more complete partial feasible solutions. For example, in a TSP, a child of a Solution object would represent its parent's partial tour, but including[excluding] one more edge (or including one more city, depending on the branching rule).

The Solution interface has the following methods:

getChildren returns a queue of the Solution objects that are the children of this Solution. The queue's retrieval order represents the application's selection rule, from most promising to least promising. In particular, the first child is cached (see Section 41.4 for an explanation of task caching).

getLowerBound returns the lower bound on the cost of any complete Solution that is an extension of this partial Solution.

getUpperBound returns an upper bound on the cost of any complete Solution, and enables an upper bound heuristic for incomplete solutions.

isComplete returns true if and only if the partial Solution is, in fact, complete.

reduce omit loose constraints. For example, in a TSP solution, this method may omit edges whose cost is greater than the current best solution, and therefore cannot be part of any better solution. This method returns void, and can have an empty implementation.

The classes that comprise the branch-and-bound framework—*provided by* JICOS to the application programmer—are described below:

BranchAndBound. This is a Task class, which resides in the jicos.applications.branchandbound package, whose objects represent a search node. A BranchAndBound Task either

- constructs smaller BranchAndBound tasks that correspond to its children search nodes, or
- fully searches a subtree, returning:
  - null, if it does not find a solution that is better than the currently known best solution
  - the best solution it finds, if it is better than the currently known best solution.

IntUpperBound. An object that represents the minimum cost among all known complete solutions. This class is in the jicos.services.shared package. It implements the Shared interface (for details about this interface, see the JICOS API), which defines the shared object. In this case, the shared object is an Integer that holds the current upper bound on the cost of a minimal solution. Consequently, IntUpperBound u "is newer than" IntUpperBound v when $u < v$.

MinSolution. This task is included in the jicos.services.tasks package. It is a composition class whose execute method

- receives an array of Solution objects, some of which may be null;
- returns the one whose cost is minimum, provided it is less than or equal to the current best solution. Equality is included to ensure that the minimum-cost *solution* is reported: It is not enough just to know the *cost* of the minimum-cost solution.
- From the standpoint of the JICOS system (not a consideration for application programming), the compose tasks form a tree that performs a gather operation, which, in this case, is a min operation on the cost of the Solution objects it receives. Each task in this gather tree is assigned to some task server, distributing the gather operation throughout the network of task servers. (This task is indeed executed on a task server, not a compute server—see Section 41.4.)

Q A queue of Solution objects. Using this framework, it is easy to construct a branch-and-bound computation. The JICOS web site tutorial [27] illustrates this, giving a complete code for a simple TSP branch-and-bound computation.

41.7 Experimental Results

41.7.1 The Test Environment

We ran our experiments on a Linux cluster. The cluster consists of 1 head machine, and 64 compute machines, composed of two processor types. Each machine is a dual 2.6 GHz (or 3.0 GHz) Xeon processor with 3 GB (2 GB) of PC2100 memory, two 36 GB (32 GB) SCSI-320 disks with on-board controller, and an on-board 1 GB ethernet adapter. The machines are connected via the gigabit link to one of two Asante FX5-2400 switches. Each machine is running CentOS 4 with the Linux smp kernel 2.6.9-5.0.3.ELsmp and the Java j2sdk1.4.2. Hyperthreading is enabled on most, but not all, machines.

41.7.2 The Test Problem

We ran a branch-and-bound TSP application, using kroB200 from TSPLIB, a 200 city Euclidean instance. In an attempt to ensure that the speedup could not be superlinear, we set the initial upper bound for the minimal-length tour with the optimum tour length. Consequently, each run explored exactly the same search tree: Exactly the same set of nodes is pruned regardless of the number of parallel processors used. Indeed, the problem instance decomposes into exactly 61,295 BranchAndBound tasks whose average execution time was 2.05 s, and exactly 30,647 MinSolution tasks whose average execution time was <1 ms.

41.7.3 The Measurement Process

For each experiment, an HSP was launched, followed by a single task server on the same machine. When additional task servers were used, they were started on dedicated machines. Each compute server was started on its own machine. Except for 28 compute servers in the 120 processor case (which were calibrated with a separate base case), each compute server thus had access to two hyperthreaded processors that are presented to the JVM as four processors (we report physical CPUs in our results). After the JICOS system was configured, a client was started on the same machine as the HSP (and task server), which executed the application. The application consists of a *deterministic workload* on a very unbalanced task graph. Measured times were recorded by JICOS's *invoice system*, which reports elapsed time (wall clock, not processor) between submission of the application's source task (aka root task) and receipt of the application's sink task's output. JICOS also automatically computes the critical path using the obvious recursive formulation for a DAG. Each test was run eight times (or more) and averages were reported.

One processor in the OS does not correspond to one physical processor. It therefore is difficult to get meaningful results for one processor. We consequently use one machine, which is two physical CPUs, as our base case. For the 120 processor measurements, we used the speedup formula of a heterogeneous processor set [28]. We thus had three separate base cases for computing the 120 processor speedup.

For our fault tolerance test, we launched a JICOS system with 32 processors as compute servers. We issued a kill command to various compute servers after 1500 s, $\sim 3/4$ through the computation. The completion time for the total computation was recorded, and was compared with the ideal completion time: $1500 + (T_{32} - 1500) \times 32/P_{final}$, where P_{final} denotes the number of compute servers that did *not* fail.

To test the overhead of running a task server on the same machine as a compute server, we ran a 22-processor experiment both with a dedicated task server and with a task server running on the same machine as one of the compute servers. We recorded the completion times and reported the averages of eight runs.

41.7.4 The Measurements

T_P denotes the time for P physical processors to run the application. A computation's *critical path time*, denoted T_∞, is a maximum time path from the source task to the sink task. We captured the critical path time for this problem instance: It is 37 s. It is well known [25] that $\max\{T_\infty, T_1/P\} \leq T_P$. Thus, $0 \leq \max\{T_\infty, T_1/P\}/T_P \leq 1$ is a *lower bound* on the fraction of perfect speedup that is actually attained.

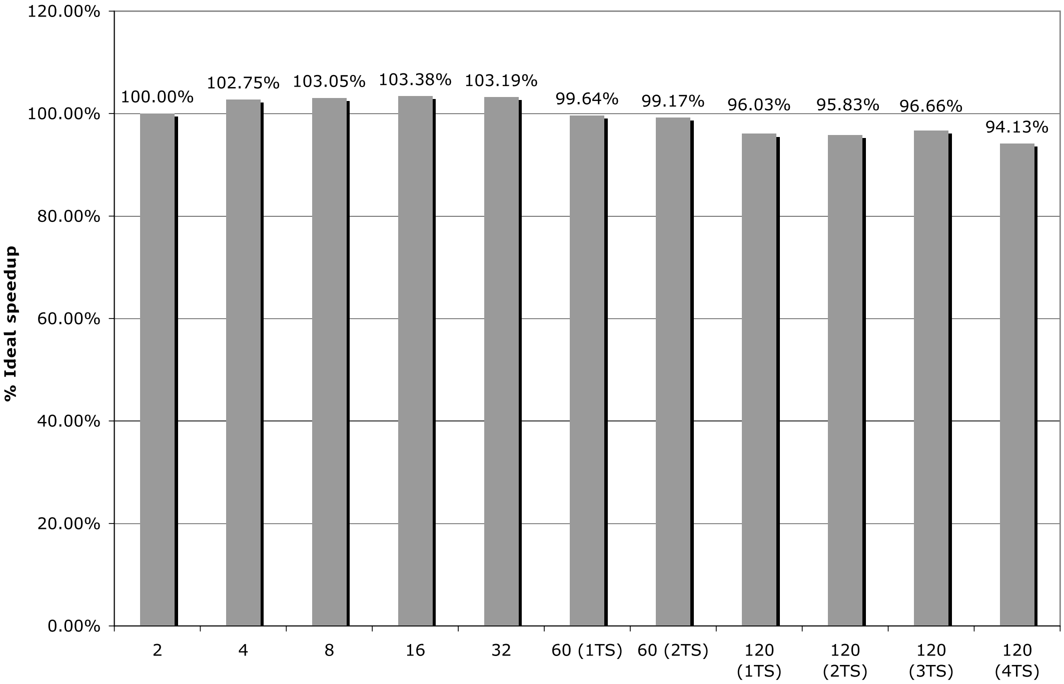


FIGURE 41.3 Number of processors versus % of ideal speedup.

Figure 41.3 presents speedup data for several experiments: The ordinate in the figure is the *lower bound* of fraction of perfect speedup. As can be seen from the figure, in all cases, the actual fraction of perfect speedup exceeds 0.94; it exceeds 0.96, when using an appropriate number of task servers. Specifically, the two-processor base case ran in 9 h and 33 min; whereas the 120-processor experiment (two processors per host) ran in just 11 min!

We get superlinear speedups for 4, 8, 16, and 32 processors. The standard deviation was <1.6% of the size of the average. As such, the superlinearity cannot be explained by statistical error. However, differences in object placement in the memory hierarchy can have impacts greater than the gap in speedup we observe [29]. So, within experimental factors beyond our control, JICOS performs well.

We are very encouraged by these measurements, especially considering the small average task times. Javelin, for example, was not able to achieve such good speedups for 2-s tasks. Even CX [28,30] is not capable of such fine task granularity.

$P_\infty = T_1 / T_\infty$ is a lower bound on the number of processors necessary to extract the *maximum parallelism* from the problem. For this problem instance, $P_\infty = 1857$ processors. Thus, 1857 processors is a lower bound on the number of processors necessary to bring the completion time down to T_∞, namely, 37 s.

Our fault tolerance data is summarized in Table 41.1. Overhead is caused by the rescheduling of tasks lost when a compute server failed as well as some time taken by the TaskServer to recognize a faulty compute server. Negative overhead is a consequence of network traffic and thread scheduling preventing a timely transfer of the kill command to the appropriate compute server.

When measuring the overhead of running a task server on a machine shared with a compute server, we received an average of 3115.1 s for a dedicated task server and 3114.8 s for the shared case. Both of these represent 99.7% ideal speedup. This is not too surprising: there is a slight reduction in communication latency having the task server on the same machine as a compute server, and the computational load of the task server is small due to the simplicity of the compose task (it is a comparison of two upper bounds). It, therefore, appears beneficial to place a compute server on every available computer in a JICOS system without dedicating machines to task servers.

TABLE 41.1 Efficiency of Compute Server Fault Tolerance

Processors (final)	30	26	12	8	6	4
Theoretical time (s)	2119.43	2214.73	3048.58	3822.87	4597.16	6145.74
Measured time (s)	2194.95	2300.92	2974.35	4182.62	4884.86	6559.91
Percent overhead	3.6	3.9	−2.4	9.4	6.3	6.7

Note: Each experiment started with 32 processors. The experiment in which 30 processors finished had 2 fail; the experiment in which 4 finished had 28 fail.

41.8 Conclusion

We have presented a framework, based on the JICOS API, for developing distributed branch-and-bound computations. The framework allows the application developer to focus on the problem-specific aspects of branch-and-bound: the lower bound computation, the upper bound computation, and the branching rule. Reducing the code required to these problem-specific components reduces the likelihood of programming errors, especially those associated with distributed computing, such as threading errors, communication protocols, and detecting, and recovering from, faulty compute servers.

The resulting application can be deployed as a distributed computation via JICOS running, for example, on a beowulf cluster. JICOS [31] scales efficiently as indicated by our speedup experiments. This, we believe, is because we have carefully provided (1) for divide-and-conquer computation; (2) an environment that is common to all compute servers for computation input (e.g., a TSP distance data structure, thereby reducing task descriptors) and a mutable shared object that can be used to communicate upper bounds as they are discovered; (3) latency hiding techniques of task caching and prefetching; and (4) latency reduction by distributing termination detection on the network of task servers.

Faulty compute servers are tolerated with high efficiency, both when faults occur (as indicated by our fault tolerance performance experiments) and when they do not (as indicated by our speedup experiments, in which no faults occur). Finally, the overhead of task servers is shown to be quite small, further confirming the efficiency of JICOS as a distributed system.

The vast majority of the code concerns JICOS, the distributed system of fault-tolerant compute servers. The Java classes comprising the branch-and-bound framework are few, and easily enhanced, or added to, by operations researchers; the source code is cleanly designed and freely available for download from the JICOS web site [27]. Our branch-and-bound framework may be used for any divide-and-conquer computation. JICOS may be adapted to solve in a distributed environment any algorithm that can be defined as a computation over directed acyclic graph, where the nodes refer to computations and the edges specify a precedence relation between computations.

References

[1] Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[2] Held, M., Hoffman, A. J., Johnson, E. L., and Wolfe, P., Aspects of the traveling salesman problem, *IBM J. Res. Dev.*, 28(4), 476, 1984.

[3] Little, J. D. C., Murty, K. G., Sweeney, D. W., and Karel, C., An algorithm for the traveling salesman problem, *Oper. Res.*, 11, 972, 1963.

[4] Land, A. H. and Doig, A. G., An automatic method for solving discrete programming problems, *Econometrica*, 28, 497, 1960.

[5] Wah, B. W., Li, G. J., and Yu, C. F., Multiprocessing of combinatorial search problems, *IEEE Comput.*, 1985.

[6] Wah, B. W. and Yu, C. F., Stochastic modeling of branch-and-bound algorithms with best-first search, *IEEE Trans. Software Eng.*, SE-11, 922, 1985.

[7] Lai, T. H. and Sahni, S., Anomalies in parallel branch-and-bound algorithms, *CACM*, 27(6), 594, 1984.

[8] Karp, R. and Zhang, Y., A randomized parallel branch-and-bound procedure, *Proc. of STOC*, 1988, p. 290.

[9] Liu, P., Aiello, W., and Bhatt, S., An atomic model for message-passing, *Proc. Symp. on Parallel Algorithms and Architectures*, 1993, p. 154.

[10] Herley, K., Pietracaprina, A., and Pucci, G., Fast deterministic parallel branch-and-bound, *Parallel Process. Lett.*, 1999.

[11] Frederickson, G., An optimal algorithm for selection in a min-heap, *Inf. Comput.*, 104, 197, 1993.

[12] Tschöke, S., Lüling, R., and Monien, B., Solving the traveling salesman problem with a distributed branch-and-bound algorithm on a 1024 processor network, *Proc. Int. Parallel Processing Symp.*, 1995, p. 182.

[13] Yahfoufi, N. and Dowaji, S., Self-stabilizing distributed branch-and-bound algorithm, *Proc. Int. Phoenix Conf. on Computers and Communications*, 1996, p. 246.

[14] Shinano, Y., Higaki, M., and Hirabayashi, R., A generalized utility for parallel branch and bound algorithms, *Proc. Int. Symp. on Parallel and Distributed Processing*, 1995, p. 392.

[15] Aversa, R., Martino, S. D., Mazzocca, N., and Venticinque, S., Mobile agent programming for clusters with parallel skeletons, *Proc. Int. Conf. on High Performance Computing for Computational Science*, Lecture Notes in Computer Science, Springer, Berlin, 2002, p. 622.

[16] Moe, R., GRIBB: Branch-and-bound methods on the Internet, *Proc. Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science, Springer, Berlin, 2003, p. 1020.

[17] Dorta, I., Leon, C., Rodriguez, C., and Rojas, A., Parallel skeletons for divide-and-conquer and branch-and-bound techniques, *Proc. Euromicro Conf. on Parallel, Distributed and Network-Based Processing*, 2003.

[18] Neary, M. O., Phipps, A., Richman, S., and Cappello, P., Javelin 2.0: Java-based parallel computing on the Internet, *Euro-Par*, 2000, p. 1231.

[19] Neary, M. O. and Cappello, P., Advanced eager scheduling for Java-based adaptively parallel computing, *Concurrency Comput.: Pract. Exp.*, 17, 797, 2005.

[20] Iamnitchi, A. and Foster, I., A problem-specific fault-tolerance mechanism for asynchronous, distributed systems, *Proc. Int. Conf. on Parallel Processing*, 2000.

[21] Seed, R. and Sandholm, T., A Note on Globus Toolkit 3 and J2EE, Technical report, Globus, 2003.

[22] Seed, R., Sandholm, T., and Gawor, J., Globus Toolkit 3 Core—a Grid Service Container Framework, Technical report, Globus, 2003.

[23] Miller, M. S. and Drexler, K. E., Markets and computation: agoric open systems, in *The Ecology of Computation*, Huberman, B., Ed., Elsevier, Amsterdam, 1988.

[24] Arnold, K., *The Jini Specifications*, 2nd ed. Addison-Wesley, Reading, MA, 1999.

[25] Blumofe, R. D., Joerg, C. F., Kuszmaul, B. C., Leiserson, C. E., Randall, K. H., and Zhou, Y., Cilk: an efficient multithreaded runtime system, *Proc. Symp. on Principles and Practice of Parallel Programming*, 1995, p. 207.

[26] Wrzesinska, G., van Nieuwpoort, R. V., Maasen, J., Kielmann, T., and Bal, H. E., Fault-tolerant scheduling of fine-grained tasks in Grid environments, *Int. J. High Performance Comput. Appl.*, to appear.

[27] JICOS: A Java-Centric Network Computing Service, Web site, http://cs.ucsb.edu/projects/jicos.

[28] Cappello, P. and Mourloukos, D., CX: a scalable, robust network for parallel computing, *Sci. Program.*, 10(2), 159, 2001.

[29] Krintz, C. and Sherwood, T., private communication, 2005.

[30] Cappello, P. and Mourloukos, D., A scalable, robust network for parallel computing, *Proc. Joint ACM Java Grande/ISCOPE Conf.*, 2001, p. 78.

[31] Cappello, P. and Coakley, C. J., JICOS: a Java-centric network computing service, *Proc. of the Int. Conf. on Parallel and Distributed Computing and Systems*, 2005.

42
Approximations for Steiner Minimum Trees

Ding-Zhu Du
University of Texas at Dallas

Weili Wu
University of Texas at Dallas

42.1 Introduction

Designing optimal network for certain purpose, such as rail road design, telecommunication, computer networks, has been an important research topic in applied mathematics and theoretical computer science for a long time. Many such problems are very difficult. Thus, finding polynomial-time approximation solutions plays an important role.

For example, given a set of points in the Euclidean plane, the shortest network interconnecting all points in the set is called the *Steiner minimum tree*. The Steiner minimum tree may contain some vertices which are not given points. Those vertices are called *Steiner points* while the points given are called *terminals*. The Steiner minimum tree for three terminals was first studied by Fermat (1601–1665). Recent research on mathematical history showed that the Steiner minimum tree for more than three points was first studied by Gauss in a letter of Gauss to Schumacher [1].

Actually, on March 19, 1836, Schumacher wrote a letter to Gauss. In this letter, he mentioned a paradox about Fermat's problem: Consider a convex quadrilateral *ABCD*. It has been known that the solution of Fermat's problem for four points A, B, C, and D is the intersection E of diagonals *AC* and BD. Suppose extending *DA* and *CB* can obtain an intersection F. Now, move A and B to F. Then E will also be moved to F. However, when the angle at F is less than 120°, the point F cannot be the solution of Fermat's problem for three given points F, D, and C. What happens (Figure 42.1)?

On March 21, 1836, Gauss wrote a letter to Schumacher in which he explained that the mistake of Schumacher's paradox occurs at the place where Fermat's problem for four points A, B, C, and D is changed to Fermat's problem for three points F, C, and D. When A and B are identical to F, the total distance from E to the four points A, B, C, and D equals $2EF + EC + ED$, not $EF + EC + ED$. Thus, the point E may not be the solution of Fermat's problem for F, C, and D. More importantly, Gauss proposed a new problem. He said that it is more interesting to find a shortest network rather than a point. Gauss also presented several possible connections of the shortest network for four given points.

Unfortunately, Gauss' letter was discovered only in 1986. From 1941 to 1986, many publications have followed Crourant and Robbins [2], who in their popular book *What Is Mathematics?* (published in 1941) referred to Gauss' problem as the Steiner tree problem.

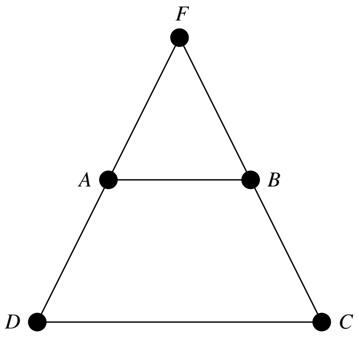


FIGURE 42.1 Schumacher's paradox.

It is well known that the Steiner minimum tree is a NP-hard problem [3,4], that is, the optimal solution is unlikely computable in polynomial time. Therefore, it is important to concentrate our research efforts seeking for good polynomial-time approximation solutions.

In 1992, Smith and Shor [5] proposed a greedy construction as follows: Start with all n terminals, regarded as a forest of n 1-node trees. At any stage, add the shortest possible line segment to the current forest, which cause two trees to merge. Continue until the forest is completely merged into a single tree. They conjectured that this greedy heuristic has a performance ratio $(3 + 2\sqrt{3})/6$, which is smaller than $\frac{2}{\sqrt{3}}$. (Here, the performance ratio for an approximation algorithm is defined to be the least upper bound for the ratio of lengths between the approximation solution and the Steiner minimum tree for the same set of terminals.) However, Du [6] disproved their conjecture by proving that this greedy heuristic has performance ratio $\frac{2}{\sqrt{3}}$. This closed the door to finding a better polynomial-time approximations for Steiner minimum trees in the Euclidean plane using this approach.

In fact, it was a long-standing open problem whether there exists a polynomial-time approximation for Steiner minimum trees in the Euclidean plane with performance ratio smaller than $\frac{2}{\sqrt{3}}$. The number $\frac{\sqrt{3}}{2}$ is the value of the Steiner ratio in the Euclidean plane. The Steiner ratio in a metric space is the largest lower bound for the ratio of lengths between the Steiner minimum tree and the minimum spanning tree for the same set of points in the space. In other words, it is the inverse of the performance ratio of the minimum spanning tree as an approximation of the Steiner minimum tree. In 1968, Gilbert and Pollak [7] conjectured that the Steiner ratio in the Euclidean plane equals $\frac{\sqrt{3}}{2}$. This conjecture received lots of attention and efforts [8–15], and was finally proved by Du and Hwang [16]; the solution received a large publicity.

In general, it was called the *better approximation problem* whether there exists a polynomial-time approximation for Steiner minimum trees in each metric space with performance ratio smaller than the inverse of the Steiner ratio.

Usually, when we talk about the Steiner trees in various metric space, we simply called the Steiner tree problem in the Euclidean plane as *the Euclidean Steiner tree*, the Steiner tree problem in the rectilinear plane as *the rectilinear Steiner tree*, and the Steiner tree problem in weighted graphs as *the network Steiner tree*. These three Steiner tree problems are classical and very well studied in the literature.

Zelikovsky [17] made the first breakthrough. He found a polynomial-time 11/6-approximation for the network Steiner tree that beats the inverse of the Steiner ratio in graphs, $\rho_2^{-1} = 2$. Soon later, Berman and Ramaiyer [18] gave a polynomial-time 92/72-approximation for the rectilinear Steiner tree and Du et al. [19] showed a general solution for the open problem. They showed that in any metric space, there exists a polynomial-time approximation with performance ratio better than the inverse of the Steiner ratio provided that for any set of a fixed number of points, the Steiner minimum tree is polynomial-time-computable. A main part of these works is to establish the lower bound for the k-Steiner ratio. Let us explain it as follows.

A tree interconnecting a terminal set is called a *Steiner tree* if every leaf is a terminal. However, a terminal in a Steiner tree may not be a leaf. A Steiner tree is *full* if every terminal is a leaf. When a terminal is not a leaf, the tree can be decomposed into several small trees at this terminal. In this way, every Steiner tree can be decomposed into smaller trees in each of which every terminal is a leaf. Those smaller trees are called *full components* of the tree. The *size* of a full component is the number of terminals in the full component.

A k*-restricted* Steiner tree is a Steiner tree with all full components of size at most k. The k-restricted Steiner minimum tree is the shortest one among all k-restricted Steiner trees. The 2-restricted Steiner minimum tree is the minimum spanning tree. The k-Steiner ratio in a metric space is the least ratio of lengths between the Steiner minimum tree and the k-restricted Steiner minimum tree for the same set of terminals in the metric space. The 2-Steiner ratio is exactly the Steiner ratio. A better lower bound for the k-Steiner ratio will give a better performance ratio for approximations of Zelikovsky's type. Zelikovsky [17] showed that the 3-Steiner ratio in graphs is at least 3/5. Du et al. [19] showed that the k-Steiner ratio in graphs is at least $\lfloor \log_2 k \rfloor / (1 + \lfloor \log_2 k \rfloor)$.

In 1996, both Arora [20] and Mitchell [21] found that actually, the Euclidean Steiner tree and the rectilinear Steiner tree have polynomial-time approximation scheme (PTAS). However, the network Steiner tree problem is found to be MAX SNP-complete, that is, unlikely to have PTAS [22].

Hwang et al. [23] gave a very useful reference. However, many important results appeared after this book was published. In this survey, we will pay attention to those important results appearing after 1990, and identify some open problems.

42.2 The Steiner Ratio

A *minimum spanning tree* on a set of terminals is the shortest network interconnecting all terminals with all edges between terminals. While the Steiner tree problem is intractable, the minimum spanning tree can be computed pretty fast. As we mentioned in the introduction, the Steiner ratio is a measure of performance of the minimum spanning tree as a polynomial-time approximation of the Steiner tree. Determining the Steiner ratio in each metric space is a traditional problem on Steiner trees. In 1976, Hwang [24] determined that the Steiner ratio in the rectilinear plane is 2/3. However, it took 22 years for completing the story of determining the Steiner ratio in the Euclidean plane. In 1968, Gilbert and Pollak conjectured that the Steiner ratio in the Euclidean plane is $\sqrt{3}/2$. Through efforts made by Pollak [25], Du et al. [8], Friedel and Widmayer [9], Booth [10], Rubinstein and Thomas [11,26], Graham and Hwang [12], Chung and Hwang [13], Du and Hwang [14], and Chung and Graham [15], the conjecture was finally proved by Du and Hwang [16,27] in 1990. The significance of their proof stems also from the potential applications of the new approach included in the proof.

In their approach, the central part is a new minimax theorem for minimizing the maximum value of several concave functions over a polytope as follows:

Minimax Theorem
Let $f(x) = \max_{i \in I} g_i(x)$, where I is a finite set and $g_i(x)$ is a continuous, concave function in a polytope X. Then the minimum value of $f(x)$ over the polytope X is achieved at some critical point, namely, a point satisfying the following property:

(*) *There exists an extreme subset Y of X such that $x \in Y$ and the index set $M(x)(= \{i \mid f(x) = g_i(x)\})$ is maximal over Y.*

The Steiner ratio problem is first transformed to such a minimax problem ($g_i(x) =$ (the length of a Steiner tree) − (the Steiner ratio)(the length of a spanning tree with graph structure i), where x is a vector whose components are edge lengths of the Steiner tree and the minimax theorem reduces the minimax problem to the problem of finding the minimax value of the concave functions at critical points. Then each critical point is transformed back to an input set of points with special geometric structure; it is a subset of a lattice formed by equilateral triangles. This special structure enables us to verify the conjecture corresponding to the nonnegativeness of minimax value of the concave functions.

Using the same approach, Gao et al. [28] proved that in any Minkowski plane (or two-dimensional Banach space), the Steiner ratio is at least 2/3. They also proved an upper bound 0.8686 for the Steiner ratio in Minkowski plane. This upper bound is very close to the conjectured upper bound $\sqrt{3}/2 = 0.8666\ldots$, which was made independently by Cieslik [29] and Du et al. [30]. Actually, many interesting open problems have a possible solution with this approach. For example, the above upper bound problem can be transformed to the following max-min problem:

$$\max_{\|\cdot\|} \min_{(A,B)} MST_{\|\cdot\|}(A, B, C) \leq \sqrt{3}$$

where the maximization is over all norm $\|\cdot\|$ in the plane, the minimization is over all possible directions of edge (A, B) in the equilateral triangle ABC with unit edges, and $MST_{\|\cdot\|}(A, B, C)$ is the length of the minimum Steiner tree for three points A, B, C in the plane with norm $\|\cdot\|$. Clearly, the Euclidean norm is the most symmetric one. However, the above minimax theorem cannot be used. One of the reasons is that possible directions of edge (A, B) form an interval which is not a finite discrete set.

The research on this minimax approach has the following three aspects:

- Develop new minimax theorems of the above type.
- Find new methods to determine critical structures.
- Generate new proof techniques for each open problem on critical structures.

Any research development on this approach would help to solve the following open problems about the Steiner ratio.

Chung–Gilbert's Conjecture on the Steiner Ratio in Euclidean Space

Steiner trees in Euclidean spaces have applications in constructing phylogenetic trees [31]. Gilbert and Pollak [7] also mentioned a possible generalization of their conjecture, that is, in any Euclidean space the Steiner ratio is achieved by the vertex set of a regular simplex. Chung and Gilbert [32] constructed a sequence of Steiner trees on regular simplexes. The lengths of constructed Steiner trees goes decreasingly to $\sqrt{3}/(4-\sqrt{2})$. Although the constructed trees are not known to be Steiner minimum trees, Chung and Gilbert conjectured that $\sqrt{3}/(4-\sqrt{2})$ is the best lower bound for Steiner ratios in Euclidean spaces. Clearly, if $\sqrt{3}/(4-\sqrt{2})$ is the limiting Steiner ratio in d-dimensional Euclidean space as d tends to infinity, then Chung–Gilbert's conjecture is a corollary of Gilbert and Pollak's generalization. However, Smith [33] and Du and Smith [34] disproved Gilbert–Pollak's general conjecture for dimension more than two. This leaves a question mark to Chung and Gilbert's conjecture. It is quite interesting that Chung–Gilbert's conjecture could still be true. In fact, the n-dimensional regular simplex may reach the smallest Steiner ratio over all sets of $n+1$ terminals in the Euclidean space. Smith and Smith [35] proved that this is true for $n = 4$.

Graham–Hwang's Conjecture on the Steiner Ratio in Rectilinear Space

Rectilinear Steiner trees in high-dimensional space can be found in biology [31,3] and optimal traffic multicasting for some communication networks [36,37]. Although the Steiner ratio in rectilinear plane was determined by Hwang [24] many years ago, there is still no progress on the Steiner ratio in rectilinear spaces, which was conjectured by Graham and Hwang [12] to be $d/(2d-1)$.

The Steiner Ratio in Banach Spaces

A Minkowski space is a Banach space of finite dimension. In other words, it is a finite-dimensional linear space with a norm. Some applications of Steiner minimum trees with L_p norm can be found in Ref. [38]. Examining the proof of Gilbert–Pollak's conjecture in the Euclidean plane, we observe that the proof has nothing to do with special properties of the Euclidean norm except the last part, verification of the conjecture on point sets with critical structure. This means that using the minimax approach to determine the Steiner ratio in Minkowski planes, we would have no problem on finding a transformation and determining critical structures. We need to work on only point sets with critical structure.

Gao et al. [28] showed that the Steiner ratios in Minkowski planes is at least 2/3. Meanwhile, Du et al. [30] showed that the Steiner ratios in Minkowski planes is at most and conjectured that this upper bound is $\sqrt{3}/2$, that is, the upper is achieved at the Euclidean norm. This conjecture is independently made by Cieslik [29].

42.3 Better Approximations

Starting from a minimum spanning tree, improve it by adding Steiner points. This is a natural idea to obtain an approximation solution for the Steiner minimum tree. Every approximation solution obtained in this way would have a performance ratio at most the inverse of the Steiner ratio. The problem is how much better than the inverse of the Steiner ratio one can make.

Over the last 20 years numerous heuristics [36,39–44] for Steiner minimum trees have been proposed for terminals in various metric spaces. Their superiority over minimum spanning trees was often claimed by computation experiments. But no theoretical proof of superiority was ever given. It was a long-standing problem whether there exists a polynomial-time approximation with performance ratio better than the inverse of the Steiner ratio or not. For simplicity, a polynomial-time approximation with performance ratio smaller than the inverse of the Steiner ratio will be called a *better* approximation.

In the following, we study two ideas to add Steiner points greedily. They give approximation algorithms with better performance ratio.

42.3.1 Chang's Idea

Chang [37,40] proposed the following approximation algorithm for Steiner minimum trees in the Euclidean plane: Start from a minimum spanning tree and at each iteration choose a Steiner point such that using this Steiner point to connect three vertices in the current tree could replace two edges in the minimum spanning tree and this replacement achieves the maximum saving among such possible replacements.

Smith et al. [45] also use the idea of the greedy improvement. But, they start with Delaunay triangulation instead of a minimum spanning tree. Since every minimum spanning tree is contained in Delaunay triangulation, the performance ratio of their approximation algorithm can also be bounded by the inverse of the Steiner ratio. The advantage of Smith–Lee–Liebman algorithm is on the running time. While Chang's algorithm runs in $O(n^3)$ time, Smith–Lee–Liebman algorithm runs only in $O(n \log n)$ time.

Kahng and Robin [46] proposed an approximation algorithm for the rectilinear Steiner tree by using the same idea as that of Chang. Approximations obtained with this idea are called *iterated 1-Steiner trees.*

42.3.2 Zelikovsky's Idea

As we mentioned in introduction, Zelikovsky's idea [17] is based on study of k-restricted Steiner trees. Clearly, for any $k \geq 3$, a k-restricted Steiner minimum tree usually has shorter length compared with a minimum spanning tree. It is natural to think about using a minimum k-restricted Steiner tree to approximate the Steiner minimum tree. However, this does not work because computing a k-restricted Steiner minimum tree is still an intractable problem. Zelikovsky's idea is to approximate the Steiner minimum tree by a 3-restricted Steiner tree generated by a polynomial-time greedy algorithm. The key fact is that the length of such an approximation is smaller than the arithmetic mean of lengths of a minimum spanning tree and a 3-restricted Steiner minimum tree; that is, the performance ratio of his approximation satisfies

$$PR \leq \frac{\rho_2^{-1} + \rho_3^{-1}}{2}$$

where ρ_k is the k-Steiner ratio. Thus, if the 3-Steiner ratio ρ_3 is higher than the Steiner ratio ρ_2, then this greedy algorithm is a better approximation for the Steiner minimum tree. Zelikovsky was able to prove that 3-Steiner ratio in graphs is at least 3/5, which is higher than 1/2, the Steiner ratio in graphs [41].

So, he solved the better approximation problem in graphs. Zelikovsky's idea has been extensively studied in the literature.

Du et al. [19] generalized Zelikovsky's idea to the k-size Steiner tree. They showed that a generalized Zelikovsky's algorithm has performance ratio

$$PR \leq \frac{(k-2)\rho_2^{-1} + \rho_k^{-1}}{k-1}$$

Berman and Ramaiyer [18] employed a different idea to generalize Zelikovsky's result. They obtained an algorithm with the performance ratio satisfying

$$PR \leq \rho_2^{-1} - \sum_{i=3}^{k} \frac{\rho_{i-1}^{-1} - \rho_i^{-1}}{i-1}$$

They also showed that in the rectilinear plane, the 3-Steiner ratio is at least 72/94, which is higher than 2/3 [24], the Steiner ratio in rectilinear plane. So, they solved the better heuristic problem in rectilinear plane.

Du et al. [19] proved a lower bound for the k-Steiner ratio in any metric space. This lower bound tends to one as k tends to infinity. So, in any metric space with the Steiner ratio less than 1, there exists a k-Steiner ratio higher than the Steiner ratio. Thus, they proved that the better heuristic exists in any metric space satisfying the following conditions:

(1) The Steiner ratio is lower than 1.
(2) The Steiner minimum tree on any fixed number of points can be computed in polynomial time.

These metric spaces include Euclidean plane and Euclidean spaces.

42.3.3 The k-Steiner Ratio ρ_k

We see in Section 42.3.2 that the determination of the k-Steiner ratio plays an important role in estimation of the performance ratio of several recent better approximations. Borchers and Du [47] completely determined the k-Steiner ratio in graphs that for $k = 2^r + h \geq 2$,

$$\rho_k = \frac{r2^r + h}{(r+1)2^r + h}$$

and Borchers et al. [48] completely determined the k-Steiner ratio in the rectilinear plane that $\rho_2 = 2/3$, $\rho_3 = 4/5$, and for $k \geq 4$, $\rho_k = (2k-1)/(2k)$. However, the k-Steiner ratio in the Euclidean plane for $k \geq 3$ is still an open problem.

42.3.4 Variable Metric Method

Berman and Ramaiyer [18] introduced an interesting approach to generalize Zelikovsky's greedy approximation. Let us call the Steiner minimum tree for a subset of k terminals as a *k-tree*. Their approach consists of two steps. The first step processes all i-trees, $3 \leq i \leq k$, sequentially in the following way: For each i-tree T with positive saving in the current graph, put T in a stack and if two leaves x and y of T are connected by a path p in a minimum spanning tree without passing any other leaf of T, then put an edge between x and y with weight equal to the length of the longest edge in p minus the saving of T. In the second step, it repeatedly pops i-trees from the stack remodifying the original minimum spanning tree for all terminals and keeping only i-trees with the current positive saving. Adding weighted edges to a point set would change the metric on the points set. Let E be an arbitrary set of weighted edges such that adding them to the input metric space makes all i-trees for $3 \leq i \leq k$ have nonpositive saving in the resulting metric space M_E. Denote by $t_k(P)$ a supermum of the length of a minimum spanning tree for the point set P in metric space M_E over all such E's. Then Berman–Ramaiyer's algorithm produces a k-size Steiner tree with total length at most

$$t_2(P) - \sum_{i=3}^{k} \frac{t_{i-1}(P) - t_i(P)}{i-1} = \frac{t_2(P)}{2} + \sum_{i=3}^{k-1} \frac{t_i(P)}{(i-1)i} + \frac{t_k(P)}{k-1}$$

The bound for the performance ratio of Berman–Ramaiyer's approximation in Section 42.3.2 is obtained from this bound and the fact that $t_k(P) \leq \rho_k^{-1} SMT(P)$, where $SMT(P)$ is the length of the Steiner minimum tree for point set P.

42.3.5 Preprocessing and Loss

Karpinski and Zelikovsky [49] proposed a preprocessing procedure to improve approximations of Zilikovsky type. First, they use this procedure to choose some Steiner points and then run a better approximation algorithm on the union of the set of terminals and the set of chosen Steiner points. This preprocessing can improve the performance ratio for many approximations that we mentioned previously.

The preprocessing procedure is similar to the algorithm of Berman and Ramaiyer. But, it uses a "related gain," instead of the saving, as the greedy potential function.

Karpinski and Zelikovsky [49] also introduced the loss of a Steiner tree, which is the length of a shortest forest connecting all Steiner points to terminals. With the loss, Hougardy and Prömel [50] defined a new greedy potential function and obtain a 1.598-approximation for the network Steiner tree. Robin and Zelikovsky [51] improved the performance ratio to 1.55. This is the best performance ratio for polynomial-time approximations of the network Steiner tree.

42.3.6 Comparison of the Two Ideas

Although Zelikovsky's idea starts from a point different from Chang's one, the two approximations are actually similar. To see this, let us describe Zelikovsky's algorithm as follows: Start from a minimum spanning tree and at each iteration choose a Steiner point such that using this Steiner point to connect three terminals could replace two edges in the minimum spanning tree and this replacement achieves the maximum saving among such possible replacements.

Clearly, they both start from a minimum spanning tree and improve it step by step by using a greedy principle to choose a Steiner point to connect a triple of vertices. The difference is only that this triple in Chang's algorithm may contain some Steiner points while it contains only terminals in Zelikovsky's algorithm. This difference makes Chang's approximation hard to be analyzed. Why? A recent paper by Du et al. [52] indicated that the saving for k-restricted Steiner trees is submodular while the saving for iterated 1-Steiner trees is not. They also analyzed the iterated 1-Steiner tree with new techniques for dealing with greedy approximations with nonsubmodular potential functions.

42.4 Approximation for Geometric Steiner Minimum Trees

In 1996, *New York Times* reported that Arora [20] obtained a surprising result: many geometric optimization problems, including the Euclidean traveling salesman and the Euclidean Steiner minimum tree, have PTAS. In this context, for any $\varepsilon > 0$, there exists a polynomial-time approximation algorithm that produces an approximation solution within a factor $1 + \varepsilon$ from optimal with run time $n^{O(1/\varepsilon)}$. Several weeks later, Mitchell [53] claimed that a "minor modification" of his earlier work [21] leads to the similar results. One year later, Arora [54] made further progress by improving run time from $n^{O(1/\varepsilon)}$ to $n^3(\log n)^{O(1/\varepsilon)}$. His new PTAS also runs randomly in time $n(\log n)^{O(1/\varepsilon)}$. Soon after, Mitchell [55] claimed again that his approach provides similar results.

Actually, both Arora's and Mitchell's approaches fall into the same category, the *adaptive partition*, consisting of a sequential partition and dynamic programming. The adaptive partition was first introduced to the design of approximation algorithm by Du et al. [56] in 1986 with guillotine cut. A sequence of guillotine cuts partitions a problem into subproblems to find an approximation solution in polynomial time by dynamic programming (see also Chapters 3 and 54).

Both Arora and Mitchell found that the cut need not be completely guillotine. In other words, the dynamic programming can still run in polynomial time if subproblems have some relations, but the number of relations should be smaller. As the number of relations goes up, the obtained approximation solution approaches to the optimal one, while the run time, of course, goes up.

To do the incomplete guillotine cut, Arora [20] employed *miportals*, while Mitchell [21,53] utilized *m-guillotine subdivision*. These two techniques differ in their applicability, that is, there exist many problems for which the portal works, but the *m*-guillotine subdivision does not, and vice versa. Actually, their combination produces further improvement of Arora [54] and Mitchell [55]. Chapter 51 discusses the application of Arora's approach to the minimum cost k-connectivity problem in geometric graphs.

42.4.1 Guillotine Cut

Roughly speaking, a *guillotine cut* is subdivision with a line segment, which divides the area into two subareas. To explain the technique of adaptive partition, let us first consider a rectangular partition problem as follows: Given a rectilinear polygon possibly with some rectangular holes, partition it into rectangles with minimum total edge length. This problem is called *minimum edge-length rectangular partition (MELRP).*

Holes in the input rectangular polygon may degenerate into a line segment or a point or partially into a line segment. The existence of holes makes difference in the computational complexity. While the MELRP in general is NP-hard, the MELRP in the hole-free case can be solved in time $O(n^4)$, where n is the number of vertices in the input rectilinear polygon.

In 1986, Du et al. [56] initiated an idea on adaptive partition with guillotine cut to design an approximation for the above rectangular partition problem. Today, we know that the idea is applicable to many geometric optimization problems, including the Euclidean Steiner minimum tree and the rectilinear Steiner minimum tree.

A cut is called a *guillotine cut* if it breaks a connected area into two parts. A rectangular partition is called a *guillotine rectangular partition* if it can be performed adaptively by a sequence of guillotine cuts. The guillotine cut features dynamic programming since each guillotine cut breaks a problem into two subproblems. Moreover, Du et al. [56] noted that the minimum length guillotine rectangular partition also satisfies the property that there exists an optimal rectangular guillotine partition in which each maximal line segment contains a vertex of the boundary. Hence, the minimum length guillotine rectangular partition can be computed by a dynamic programming in $O(n^5)$ time. Therefore, they suggested to use the minimum length guillotine rectangular partition to approximate the MELRP and tried to analyze the performance ratio. Unfortunately, they failed to get a constant ratio in general and obtained a result only in a special case. In this special case, the input is a rectangle with points inside. Those points are holes. It had been showed that even the rectangular partition case is NP-hard, and Gonzalez and Zheng [57] had introduced divide and conquer approximation algorithms for this problem.

In their proof, Du et al. [56] introduced two terms $proj_x(P)$ and $proj_y(P)$ to form a stronger inequality than the one that they want to prove. That is, instead of proving that for each rectangular partition P, by adding some segments, a guillotine rectangular partition P' can be obtained to satisfy that $length(P') \leq 2 \cdot length(P)$, they show that P' satisfies $length(P') \leq 2 \cdot length(P) - proj_x(P) - proj_y(P)$, where $proj_x(P)$ ($proj_y(P)$) denote the total length of segments on a horizontal (vertical) line covered by vertical (horizontal) projection of the partition P. Chapter 54 discusses a different proof to establish this approximation bound as well as other approximation algorithms for this problem.

The term $proj_x(P)$ ($proj_y(P)$) plays an important role in the induction. When a vertical (horizontal) guillotine cut receives horizontal (vertical) projections from both sides, a portion of $proj_x(P)$ ($proj_y(P)$) will be doubled from the application of the induction hypothesis to the two subproblems resulting from the cut. The increased negative value would cancel the increased positive value caused by the length of the new cut.

For simplicity, let us call any point in such cut a 1-*dark point*. That is, a point lying inside of a given area is a horizontal (vertical) 1-dark point if it receives horizontal (vertical) projections from both sides. What Du et al. [56] did was to find various guillotine cuts consisting of 1-dark points. They succeeded only in the mentioned special case.

42.4.2 *m*-Guillotine Subdivision

Mitchell [21] made an important discovery about 1-dark points.

Lemma 42.1 **(Mitchell's Lemma)**

Let H (V) be the set of all horizontal (vertical) 1*-dark points. Then there exists either a horizontal line L such that*

$$length(L \cap H) \leq length(L \cap V)$$

or a vertical line L such that

$$length(L \cap H) \geq length(L \cap V)$$

The first inequality means that if we use all horizontal 1-dark points on the horizontal line L to form an incomplete guillotine cut, then the set of vertical 1-dark points on L will receive enough two-side projection to cancel the length of the cut. Such cut is called a horizontal 1-*guillotine cut*. Similarly, the set of all vertical 1-dark points on a vertical line forms a vertical 1-*guillotine cut*. The lemma actually says that either an expected horizontal 1-guillotine cut or an expected vertical 1-guillotine cut exists.

A rectangular partition is called a 1-*guillotine rectangular partition* if it can be performed by a sequence of 1-guillotine cuts. It has been shown that there exists minimum 1-guillotine rectangular partition such that every maximal segment contains at least a vertex of the boundary.

Now, the question is whether the 1-guillotine cut can also feature the dynamic programming. The answer is yes, although the 1-guillotine cut partitions a rectangular partition problem into two subproblems with some boundary conditions that two open segments may be created on the boundary by a 1-guillotine cut. This boundary condition increases the number of subproblems in the dynamic programming. Since, each subproblem is based on a rectangle with four sides, the condition on each side can be described by two possible open segments at two ends. Hence each side has $O(n^2)$ possible conditions. So, the total number of possible boundary conditions is $O(n^8)$. Thus the total number of possible subproblems is $O(n^{12})$. For each subproblem, there are $O(n^3)$ possible 1-guillotine cuts. Therefore, the minimum 1-guillotine rectangular partition can be computed by dynamic programming in $O(n^{15})$ time. With 1-guillotine cuts, the approximation ratio 2 can be established not only for the special case, but also in general. However, to reduce the number of boundary conditions in the general case, Mitchell [21] initially covered the input by a rectangle and kept performing a "rectangular partition" at each iteration.

Mitchell [21] also presented the proof in a different way. Instead of introducing the terms $proj_x$ and $proj_y$, he did a symmetric charge. For example, in case when a horizontal 1-guillotine cut is used, charge 0.5 to all horizontal segments whose projection directly contributes horizontal 1-dark points on the cut. Since the charge is performed symmetrically, no segment in P can be charged more than twice on the same side. Therefore, each segment in P can be charged with at most value 1 and the total length of charged segments cannot exceed the total length of P. This argument has the same power as that of using projection, however, more directly to the point.

The 1-guillotine cut can be easily generalized in the following way: A point p is a horizontal (vertical) *m-dark point* if the horizontal (vertical) line passing through p intersects at least $2m$ vertical (horizontal) segments of the considered rectangular partition P, among which at least m are on the left of p (above p) and at least m are on the right of p (below p). A horizontal (vertical) cut is an *m-guillotine cut* if it consists of all horizontal (vertical) m-dark points on the cut line. In other words, let H_m (V_m) denote the set of all horizontal (vertical) m-dark points. An m-guillotine cut is either a horizontal line L with cut segment

$$L \cap H_m \subseteq L \cap P$$

or a vertical line L with cut segment

$$L \cap V_m \subseteq L \cap P$$

where P is a considered partition.

A rectangular partition is m-guillotine if it can be realized by a sequence of m-guillotine cuts. The minimum m-guillotine rectangular partition can also be computed by dynamic programming in $O(n^{10m+5})$ time. In fact, at each step, an m-guillotine cut has at most $O(n^{2m+1)})$ choices. There are $O(n^4)$ possible rectangles appearing in the algorithm. Each rectangle has $O(n^{8m})$ possible boundary conditions. Mitchell's lemma can also be generalized to the following.

Lemma 42.2 **(Mitchell's Lemma)**

There exists either a horizontal line L such that

$$length(L \cap H_m) \leq length(L \cap V_m)$$

or a vertical line L such that

$$length(L \cap H_m) \geq length(L \cap V_m)$$

With the m-guillotine subdivision, Mitchell [53] showed that there exists a polynomial-time $(1 + \varepsilon)$-approximation with run time $n^{O(\log 1/\varepsilon)}$ for the rectangular partition.

To apply the m-guillotine subdivision to the Steiner tree problem, we only need to include all terminals in a rectangle at the initial step.

42.4.3 Portals

Arora's PTAS [20] is also based on a sequential rectangular partition. Consider the rectilinear Steiner minimum tree. Initially, use a minimal square to cover n input terminals. Then with a sequence of cuts, partition the square into small rectangles, each containing one terminal. Arora [20] managed the partition to have $O(\log n)$ levels with the following techniques:

(1) Equally divide the initial square into $n^2 \times n^2$ grid. Move each input point to the center of the cell containing the input point.
(2) Choose a grid line in a range between 1/3 and 2/3 of the longest edge to cut.

Let P be the set of n input points and P' the set of n' cell-centers receiving the n input points in (1). (Possibly, $n' < n$.) Then, it is shown that there is a PTAS for P if and only if there exists a PTAS for P'. Therefore, one may work on P' instead of P.

Furthermore, with technique (2), the rectangle at the ith level has area at most $S^2(2/3)^{i-1}$, where S is the edge length of the initial square. Since the ratio between the lengths of longer edge and shorter edge is at most 3, the rectangle at the last level, say the sth level, has area at least $(1/3)(S/n^2)^2$. Therefore, $S^2(2/3)^{s-1} \geq (1/3)(S/n^2)^2$. That is, $s = O(\log n)$.

To reduce the number of crosspoints at each cut line, Arora employed a technique, called the *portal*. Portals are points on cut line equally dividing cut segments. For rectilinear Steiner tree, crosspoints on a cut line can be moved to portals by increasing a small amount of the length. This would decrease the number of crosspoints on the cut line. Suppose the number of portals is p, and we will call such portals as *p-portals*. By properly choosing cut line, at each level of the adaptive partition, moving crosspoints to portals would increase the length of the tree within three pth of the total length of the Steiner tree. To see this, consider each rectangle R at a certain level. Suppose its longer edge has length a and shorter edge has length b $(a/3 \leq b \leq a)$. Look at every possible cut in a range between 1/3 and 2/3 of a longer edge. Choose the cut line to intersect the Steiner tree with the smallest number of crosspoints, say c points. Then the length of the part of the Steiner tree lying in rectangle R is at least $ca/3$. Moving c crosspoints to portals requires to add some segments with total length at most $cb/(p+1) \leq ca/(p+1) \leq (3/p)(ca/3)$.

Since the guillotine rectangular partition has $O(\log n)$ levels, the total length of the resulting Steiner tree is within $(1 + \frac{3}{p})^{O(\log n)}$ times the length of the optimal one. To obtain $(1 + \frac{3}{p})^{O(\log n)} \leq 1 + \varepsilon$, we have to choose $p = O(\frac{\log n}{\varepsilon})$.

Summarizing the above, we obtained the structure theorem for the rectilinear Steiner minimum tree.

Theorem 42.1

For rectilinear SMT, there exists a $(1 + \varepsilon)$-approximation T together with a guillotine rectangular partition P such that each cut intersects T only at p-portals where $p = O(\frac{\log n}{\varepsilon})$.

Now, one can employ dynamic programming to find the shortest one among $(1 + \varepsilon)$-approximations described in the structure theorem. To estimate the run time of dynamic programming, note that each subproblem is characterized by a rectangle and conditions on the boundary. There are $O(n^8)$ possible rectangles. Each rectangle has four sides. One of them must contain p portals. However, each of the other three may contain less than p portals resulting from previous cuts. There are $O(n^4)$ possible previous cuts for each edge. Thus, the number of possible sets of positions for portals on each of these three sides is $O(n^4)$. Hence, the total number of possible sets of portal positions on the boundary is $O(n^{20})$. For each fixed set of portal positions, we need also consider whether a portal is a crosspoint or not and how crosspoints are connected with each other inside the rectangle. It brings us $2^{O(p)}$ possibilities. Therefore, the total number of possible subproblems is $n^{O(1/\varepsilon)}$. Moreover, in each iteration of dynamic programming, the number of all possible cuts is $O(n^2)$. Therefore, the dynamic programming runs in time $n^{O(1/\varepsilon)}$. the dynamic programming runs in time $n^{O(1/\varepsilon)}$.

42.4.4 Comparison and Combination

Let us compare two techniques, the m-guillotine subdivision and the portal.

For problems in d-dimensional space, the cut line should be replaced by $(d - 1)$-dimensional hyperplane. The number of portals would be $O((m \log n)^{d-1})$ where $m = 1/\varepsilon$. With so many possiblecrosspoints, the dynamic programming runs in time $2^{(m \log n)^{d-1}}$, which is not a polynomial for $d \geq 3$. However, the m-guillotine cut has $O(m)$ crosspoints in each dimension and totally $O(m^{d-1})$ crosspoints in $(d - 1)$-dimensional cut plane. Therefore, the dynamic programming runs in time $n^{O(m^{d-1})}$, which is still polynomial for fixed m. Thus, while the m-guillotine works well in high-dimensional space, the portal does not.

The portal technique cannot apply to the above rectangular partition problems and the rectilinear Steiner arborescence [58]. In fact, for these three problems, moving crosspoints to portals is sometimes impossible. But, the m-guillotine cut works well in these problems.

However, the portal preserves the topological structure of original figure, but, the m-guillotine cut does not. Therefore, for problems such as the Euclidean k-medians and the Euclidean facility locations [59], the portal works well, but the m-guillotine cut does not.

In case when the problem is suitable to use both techniques, such as the rectilinear Steiner minimum tree and the Euclidean Steiner tree, a rough idea suggests a combination of two techniques, which may reduce the run time. In fact, one may first move crosspoints to $O(m \log n)$ portals and then choose $2m$ from the $O(m \log n)$ portals to construct an m-guillotine cut ($m = 1/\varepsilon$). This way, the dynamic programming will run in time $n^c(\log n)^{O(m)})$ for some constant c. However, the problem is that in two techniques, two different principles are used for choosing the position of each cut. In m-guillotine cut, the cut line has to satisfy the inequality in Mitchell's lemma. But, when portals are used, the cut line is chosen to minimize the number of crosspoints.

Again, Arora [54] and Mitchell [55] found two different ways to overcome this problem.

Arora [54] borrowed a shafting technique from nonadaptive partition and employed random argument. He found family of n^2 adaptive partitions obtained by shafting, satisfying property that the total increased length caused by moving crosspoints to portal and m-guillotine cutting in average bounded by $\varepsilon \cdot OPT$.

Mitchell [55] tried another way. First, he employed a 1-guillotine cut and put portals on the 1-guillotine cut segment; then he constructed an m-guillotine cut with the portals. The purpose of putting portals on the 1-guillotine cut is to bound the length increased by moving crosspoints to portals by the length of the 1-guillotine cut segment. A new lemma of Mitchell's type on the combination of the 1-guillotine cut and the m-guillotine cut would resolve the problem.

It is interesting to mention that combining Arora's and Mitchell's approaches of 1997 again may produce a further improvement in the run time [60].

Rao and Smith [61] reduced time complexity of PTAS to $m^{O(m)} n \log n$ for the rectilinear Steiner tree and the Euclidean Steiner tree. The basic idea is to compute a *banyan* instead of the tree. The banyan contains a $(1 + \varepsilon)$-approximation solution. It has $O(n)$ vertices and the total length is within a constant factor from optimal solution. Moreover, it can be computed in time $O(n \log n)$.

References

[1] Gauss, C. F., Briefwechsel Gauss-Schuhmacher, in *Werke Bd. X, 1*, Kgl. Gesellschaft der Wissenschaften, Göttingen, 459, 1917.

[2] Crourant, R. and Robbins, H., *What Is Mathematics?* Oxford University Press, New York, 1941.

[3] Foulds, L. R. and Graham, R. L., The Steiner problem in Phylogeny is NP-complete, *Adv. Appl. Math.*, 3, 43, 1982.

[4] Garey, M. R., Graham, R. L., and Johnson, D. S., The complexity of computing Steiner minimal trees, *SIAM J. Appl. Math.*, 32, 835, 1977.

[5] Smith, W. D. and Shor, P. W., Steiner tree problems, *Algorithmica*, 7, 329, 1992.

[6] Du, D.-Z., On greedy heuristics for Steiner minimum trees, *Algorithmica*, 13, 381, 1995.

[7] Gilbert, E. N. and Pollak, H. O., Steiner minimal trees, *SIAM J. Appl. Math.*, 16, 1, 1968.

[8] Du, D.-Z., Hwang, F. K., and Yao, E. Y., The Steiner ratio conjecture is true for five points, *J. Combin. Theor., Ser. A*, 38, 230, 1985.

[9] Friedel, J. and Widmayer, P., A simple proof of the Steiner ratio conjecture for five points, *SIAM J. Appl. Math.*, 49, 960, 1989.

[10] Booth, R. S., The Steiner ratio for five points, *Disc. Comput. Geom.*, 1991.

[11] Rubinstein, J. H. and Thomas, D. A., A variational approach to the Steiner network problem, *Proc. NATO Workshop on Topological Networks*, Copenhagen, 1989.

[12] Graham, R. L. and Hwang, F. K., Remarks on Steiner minimal trees, *Bull. Inst. Math. Acad. Sinica*, 4, 177, 1976.

[13] Chung, F. R. K. and Hwang, F. K., A lower bound for the Steiner tree problem, *SIAM J. Appl. Math.*, 34, 27, 1978.

[14] Du, D.-Z. and Hwang, F. K., A new bound for the Steiner ratio, *Trans. Am. Math. Soc.*, 278, 137, 1983.

[15] Chung, F. R. K. and Graham, R. L., A new bound for Euclidean Steiner minimum trees, *Ann. N.Y. Acad. Sci.*, 440, 328, 1985.

[16] Du, D.-Z. and Hwang, F. K., The Steiner ratio conjecture of Gilbert–Pollak is true, *Proc. Nat. Acad. Sci.*, Vol. 87, 1990, p. 9464.

[17] Zelikovsky, A. Z., The 11/6-approximation algorithm for the Steiner problem on networks, *Algorithmica*, 9, 463, 1993.

[18] Berman, P. and Ramaiyer, V., Improved approximations for the Steiner tree problem, *J. Algorithms*, 17, 381, 1994.

[19] Du, D.-Z., Zhang, Y., and Feng, Q., On better heuristic for Euclidean Steiner minimum trees, *Proc. FOCS*, 1991, pp. 431–439.

[20] Arora, S., Polynomial-time approximation schemes for Euclidean TSP and other geometric problems, *Proc. FOCS*, 1996, p. 2.

[21] Mitchell, J. S. B., Guillotine subdivisions approximate polygonal subdivisions: a simple new method for the geometric k-MST problem, *Proc. SODA*, 1996, p. 402.

[22] Bern, M. and Plassmann, P., The Steiner problem with edge lengths 1 and 2, *Inf. Proc. Lett.*, 32, 171, 1989.

[23] Hwang, F. K., Richards, D. S., and Winter, P., *Steiner Tree Problems*, North-Holland, Amsterdam, 1992.

[24] Hwang, F. K., On Steiner minimal trees with rectilinear distance, *SIAM J. Appl. Math.*, 30, 104, 1976.

[25] Pollak, H. O., Some remarks on the Steiner problem, *J. Combin. Theor., Ser. A*, 24, 278, 1978.

[26] Rubinstein, J. H. and Thomas, D. A., The Steiner ratio conjecture for six points, *J. Combin. Theor., Ser. A*, 58, 54, 1991.

[27] Du, D.-Z. and Hwang, F. K., An approach for proving lower bounds: solution of Gilbert–Pollak's conjecture on Steiner ratio, *Proc. FOCS*, 1990, p. 76.

[28] Gao, B., Du, D.-Z., and Graham, R. L., A tight lower bound for the Steiner ratio in Minkowski planes, *Disc. Math.*, 142(1), 49–63, 1995.

[29] Cieslik, D., The Steiner ratio of Banach–Minkowski planes, in *Contemporary Methods in Graph Theory*, Bodendiek, R., Ed., BI-Wissenschatteverlag, Mannheim, 1990, p. 231.

[30] Du, D.-Z., Gao, B., Graham, R. L., Liu, Z.-C., and Wan, P.-J., Minimum Steiner trees in normed planes, *Disc. Comput. Geom.*, 9, 351, 1993.

[31] Cavalli-Sforza, L. L. and Edwards, A. W., Phylogenetic analysis: models and estimation procedures, *Am. J. Hum. Genet.*, 19, 233, 1967.

[32] Chung, F. R. K. and Gilbert, E. N., Steiner trees for the regular simplex, *Bull. Inst. Math. Acad. Sinica*, 4, 313, 1976.

[33] Smith, W. D., How to find Steiner minimal trees in Euclidean d-space? *Algorithmica*, 1991.

[34] Du, D.-Z. and Smith, W. D., Three disproofs for Gilbert–Pollak conjecture in high dimensional spaces, *J. Combin. Theor.*, 74, 115–130, 1996.

[35] Smith, W. D. and Smith, J. M., On the Steiner ratio in 3-space, *J. Comb. Theor. Ser. A*, 1995.

[36] Bharath-Kumar, K. and Jaffe, J. M., Routing to multiple destinations in computer networks, *IEEE Trans. Comm.*, COM-31, 343, 1983.

[37] Chang, S.-K., The design of network configurations with linear or piecewise linear cost functions, *Symp. on Comp., Comm., Networks, and Teletraffic*, 1972, 363.

[38] Smith, J. M. and Liebman, J. S., Steiner trees, Steiner circuits and interference problem in building design, *Engin. Opt.*, 4, 15, 1979.

[39] Beasley, J. E., A Heuristic for the Euclidean and Rectilinear Steiner Problems, Technical report of the Management School, Imperial College, London, 1989.

[40] Chang, S.-K., The generation of minimal trees with a Steiner topology, *JACM*, 19, 699, 1972.

[41] Karp, R. M., Reducibility among combinatorial problems, in *Complexity of Computer Computation*, Miller, R. E. and Tatcher, J. W., Eds., Plenum Press, New York, 1972, p. 85.

[42] Kou, L. and Makki, K., An even faster approximation algorithm for the Steiner tree problem in graphs, *Congressus Numerantium,* 59, 1987, 147.

[43] Kou, L., Markowsky, G., and Berman, L., A fast algorithm for Steiner trees, *Acta Inform.*, 15, 141, 1981.

[44] Kuhn, H. W., Steiner's problem revisited, in *Studies in Optimization*, Dantzig, G. B. and Eaves, B. C., Eds., MAA, Washington, DC, 1974, pp. 52–70.

[45] Smith, J. M., Lee, D. T., and Liebman, J. S., An $O(N \log N)$ heuristic for Steiner minimal tree problems in the Euclidean metric, *Networks*, 11, 23, 1981.

[46] Kahng, A. and Robins, G., A new family of Steiner tree heuristics with good performance: The iterated 1-Steiner approach, *Proc. ICCAD*, 1990, p. 428.

[47] Borchers, A. and Du, D.-Z., The k-Steiner ratio in graphs, *SIAM J. Comput.*, 26(3), 857–869, 1997.

[48] Borchers, A., Du, D.-Z., Gao, B., and Wan, P.-J., The k-Steiner ratio in the rectilinear plane, *J. Algorithms*, 29, 1, 1998.

[49] Karpinski, M. and Zelikovsky, A. Z., A new approach to approximation of Steiner trees, *Journal of Combinatorial Optimization,* 1, 47–65, 1997.

[50] Hougardy, S. and Prömel, H. J., A 1.598-approximation algorithm for the Steiner problem in graphs, *Proc. SODA*, 1999, p. 448.

[51] Robin, G. and Zelikovsky, A., Improved Steiner trees approximation in graphs, *Proc. SODA*, 2000, p. 770.

[52] Du, D.-Z., Graham, R. L., Wu, W., Pardalos, P., and Wan, P.-J., Analysis of greedy approximations with nonsubmodular potential functions, 2006 (unpublished manuscript).

[53] Mitchell, J. S. B., Guillotine subdivisions approximate polygonal subdivisions: Part II—a simple polynomial-time approximation scheme for geometric k-MST, TSP, and related problem, *SIAM J. Comput.*, 29, 515, 1999.

[54] Arora, S., Nearly linear time approximation schemes for Euclidean TSP and other geometric problems, *Proc. of FOCS*, 1997, p. 554.

[55] Mitchell, J. S. B., Guillotine subdivisions approximate polygonal subdivisions: Part III—faster polynomial-time approximation scheme for geometric network optimization, *Proc. Canadian Conf. on Comput. Geom.*, 1997, p. 229.

[56] Du, D.-Z., Pan, L.-Q., and Shing, M.-T., Minimum edge length guillotine rectangular partition, Technical report 0241886, Mathematical Sciences Research Institute, University of California, Berkeley, 1986.

[57] Gonzalez, T. F. and Zheng, S. Q., Bounds for partitioning rectilinear polygons, *Proc. Symp. on Comput. Geom.*, 1985, pp. 281–287.

[58] Lu, B. and Ruan, L., Polynomial time approximation scheme for the rectilinear Steiner arborescence problem, *J. Comb. Opt.*, 4, 357, 2000.

[59] Arora, S., Raghavan, P., and Rao, S., Polynomial time approximation schemes for Euclidean k-medians and related problems, *Proc. STOC*, 1998, p. 106.

[60] Arora, S., Polynomial-time approximation schemes for Euclidean TSP and other geometric problems, *JACM*, 45, 753, 1998.

[61] Rao, S. B. and Smith, W. D., Approximating geometrical graphs via "spanners" and "banyans," *Proc. STOC*, 1998, p. 540.

43

Practical Approximations of Steiner Trees in Uniform Orientation Metrics

Andrew B. Kahng
University of California at San Diego

Ion Măndoiu
University of Connecticut

Alexander Zelikovsky
Georgia State University

43.1 Introduction

The Steiner minimum tree problem, which asks for a minimum-length interconnection of a given set of terminals in the plane, is one of the fundamental problems in very large-scale integration (VLSI) physical design. Although advances in VLSI manufacturing technologies have introduced additional routing objectives, minimum length continues to be the primary objective when routing noncritical nets, since the minimum-length interconnection has minimum total capacitance and occupies minimum amount of area.

To simplify design and manufacturing, VLSI interconnect is restricted to a small number of orientations defining the so-called *interconnect architecture*. Until recently, designers have relied almost exclusively on the *Manhattan interconnect architecture*, which allows interconnect routing along two orthogonal directions. However, non-Manhattan interconnect architectures—such as the *Y-architecture*, which allows 0°, 120°, and 240° oriented wires, and the *X-architecture*, which allows 45° diagonal wires in addition to the traditional horizontal and vertical orientations—are becoming increasingly attractive because of the significant potential for reducing interconnect length (see, e.g., Refs. [1–7]). A common generalization of interconnect architectures of interest in VLSI design is that of *uniform orientation metric*, or *λ-geometry*, in which routing is allowed only along $\lambda \geq 2$ orientations forming consecutive angles of π/λ. The Manhattan, Y-, and X-architectures correspond to $\lambda = 2$, 3, and 4, respectively.

In contrast to the extensive literature on the rectilinear version of the Steiner tree problem, computing Steiner trees in λ-geometries with $\lambda > 2$ has received much less attention in the literature. A hierarchical Steiner tree construction was proposed by Sarrafzadeh and Wong [8]. Koh [9] showed how to compute the optimal Steiner trees for three terminals under the octilinear metric ($\lambda = 4$) and proposed a heuristic inspired by the iterated 1-Steiner heuristic of Kahng and Robins for computing rectilinear Steiner tree. Li et al. [10] solved the 3-terminal case for $\lambda = 3$ and proposed a simulated annealing heuristic. Generalizations of the classical Hanan grid [11] for $\lambda = 3$ and $\lambda = 4$ are also proposed by Refs. [9,10], but these generalizations lead to multilevel grids typically containing too many points to be of use in designing practical algorithms.

Chiang et al. [12] proposed a simple heuristic for constructing octilinear Steiner trees that computes a rectilinear Steiner tree for the terminals and then iteratively reduces tree length by using diagonal wires to reroute tree edges and by sliding Steiner points. Coulston [13] proposed an exact octilinear Steiner tree algorithm based on a two-phase approach of generating all possible full components for the given set of terminals and then merging them to form an optimal tree. He reports that the proposed algorithm has practical runtime for up to 25 terminals. Nielsen et al. [14] described the extension of the GeoSteiner exact rectilinear/Euclidean Steiner tree package to arbitrary λ-geometries. GeoSteiner uses the same two-phase approach as Coulston's algorithm, however it incorporates highly effective pruning techniques based on structural properties of full components to greatly improve the running time. With these improvements, GeoSteiner is reported to compute optimal λ-geometry Steiner trees for instances with 100 random terminals in seconds of CPU time.

Recently, there has been a growing interest in practical methods for computing near-optimal Steiner trees for instances with up to *tens of thousands of terminals*. Instances of this size, for example, scan enable nets, are becoming more common in modern VLSI designs because of the increased emphasis on design for test. Such nets are noncritical for chip performance and tend to consume significant routing resources, so minimizing length is the appropriate optimization objective. Furthermore, very large Steiner tree instances are created by reductions that model nonzero terminal dimensions, for example, nets with preroutes. High-quality routing of such instances requires representing each terminal by a set of electrically equivalent points [15] and this results in instances with as much as 100,000 points [16]. Due to combinatorial explosion and/or quadratic memory requirements, instances of this size cannot be solved in practical time with the existing exact methods such as GeoSteiner or best-performing heuristics such as iterated 1-Steiner [17] and iterated Rajagopalan-Vazirani (IRV) [18].

In this chapter we present a highly scalable heuristic for computing near-optimal Steiner trees. Our heuristic, referred to as the *batched greedy algorithm* (BGA) in the following, is graph-based and can therefore be easily modified to handle routing in uniform orientation geometries as well as other practical considerations such as routing obstacles, preferred directions [19], and via costs. Indeed, the results reported in Section 43.5 show only a small factor increase in runtime compared to the rectilinear implementation. The BGA heuristic routes a 34k-terminals net extracted from a real design in less than 25 s compared to over 86 min needed by the $O(n^2)$ edge-based heuristic of Ref. [20]. More importantly, this dramatic reduction in runtime is achieved with no loss in solution quality. On random instances with more than 100 terminals our algorithm improves over the rectilinear minimum spanning tree (MST) by an average of 11%, matching in solution quality the edge-based heuristic of Ref. [20].

The BGA heuristic derives its efficiency from three key ideas:

- Combining the implementation proposed in Ref. [21] for the *greedy triple contraction algorithm* (GTCA) of Zelikovsky [22] with the batched method introduced by the iterated 1-Steiner heuristic of Kahng and Robins [17].
- A new divide-and-conquer method for computing a superset of size $O(n \log n)$ of the set of $O(n)$ triples required by GTCA.
- A new linear size data structure that enables finding a bottleneck (i.e., maximum cost) edge on the tree path between two given nodes in $O(\log n)$ time after $O(n \log n)$ preprocessing, with very low constants hidden under the big O. This data structure allows computing the gain of a triple (see Section 43.2 for the definition) in $O(\log n)$ time.[1]

The BGA heuristic requires $O(n)$ memory and $O(n \log^2 n)$ time for computing a Steiner tree over n terminals. To the best of our knowledge, this is the first practical subquadratic heuristic with such

[1] Our data structure may be of interest in other applications that require computing bottleneck edges. For example, Zachariasen incorporated it in the beta version of the GeoSteiner 4.0 code for computing optimum geometric Steiner trees. On large instances, computing bottleneck edges with the new data structure was found to be faster than look up in a precomputed matrix, most likely because of improved memory access locality [23].

high performance. The $O(n \log n)$ implementation described in Ref. [20] for the edge-based heuristic requires advanced data structures, potentially involving large hidden constants. We are not aware of any implementation demonstrating the practical applicability of this implementation. After the publication of a preliminary version of this work [24], Zhu et al. [25] proposed an $O(n \log n)$ octilinear Steiner tree heuristic based on spanning graphs, which is reported to run faster than BGA at the cost of a small decrease in solution quality.

The rest of the chapter is organized as follows. In Section 43.2 we briefly review the GTCA of Ref. [22] and describe our new BGA. In the following two sections we describe in detail two of the key BGA subroutines: In Section 43.3 we give the new divide-and-conquer method for computing the set of $O(n \log n)$ triples used by BGA, while in Section 43.4 we describe the new data structure for computing bottleneck edges. Finally, in Section 43.5 we give experimental results comparing BGA with previous implementations of rectilinear and octilinear Steiner tree heuristics and exact algorithms on test cases both randomly generated and extracted from recent VLSI designs.

43.2 The Greedy Triple Contraction and Batched Greedy Algorithms

We begin this section by introducing the Steiner tree terminology used in the rest of the chapter. A Steiner tree for a set of terminals is a tree spanning the terminals and possibly additional points, called *Steiner points*. A Steiner tree is called a *full Steiner tree* if all terminals are leaves (i.e., have degree 1). Any Steiner tree T can be split into edge-disjoint full Steiner trees called the *full Steiner components* of T [26]. A Steiner tree T is called k-restricted if every full component of T has at most k terminals (an example of a 3-restricted rectilinear Steiner tree is shown in Figure 43.1). The minimum-cost 3-restricted Steiner tree is in general cheaper than the MST of the terminals (note that the MST is the minimum-cost 2-restricted Steiner tree of the terminals).

The GTCA in Ref. [22] finds an approximate minimum-cost 3-restricted Steiner tree by greedily choosing 3-restricted full components which reduce the cost of the MST. To describe GTCA we need to introduce a few more notations. Let G_S be the complete graph on a given set S of terminals, and let $MST(S)$ be a MST of G_S. A *triple* τ is an optimal Steiner tree for a set of three terminals.[2] We denote by *center*(τ)

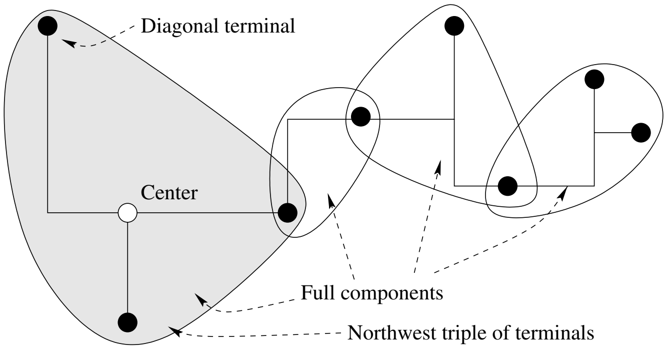


FIGURE 43.1 A 3-restricted rectilinear Steiner tree partitioned into full components. The dark full component is a northwest triple (From Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., *Proc. Asia and South Pacific Design Automation Conf.*, 2003, pp. 828–832.).

[2] The optimum Steiner tree of three given terminals can be computed in constant time under the common uniform orientation metrics, including rectilinear [11] and octilinear [3] metrics.

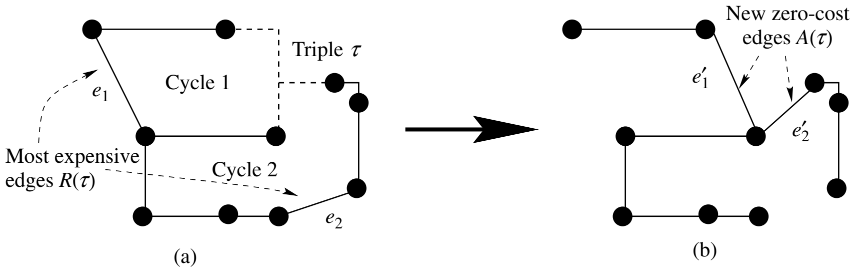


FIGURE 43.2 The MST of G_S (a) before and (b) after contraction of the triple τ. The gain of the dashed triple is the difference between the cost of most expensive edges $R(\tau) = \{e_1, e_2\}$ in each of the two cycles of $T \cup \tau$ and the cost of τ. Two new zero-cost edges $A(\tau) = \{e_1', e_2'\}$ replace e_1 and e_2 in the updated MST (From Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., *Proc. Asia and South Pacific Design Automation Conf.*, 2003, pp. 828–832.).

the single Steiner point of τ and by $cost(\tau)$ the cost of τ. In the graph $MST(S) \cup \tau$, there are two cycles (see Figure 43.2[a]). To obtain an MST of this graph we should remove the most expensive edge from each cycle. Let e_1 and e_2 be the two edges that must be removed and let $R(\tau) = \{e_1, e_2\}$. The *gain* of τ is $gain(\tau) = cost(R(\tau)) - cost(\tau)$.

GTCA (see Figure 43.3) repeatedly adds a triple τ with the largest gain and *contracts* it, that is, collapses the three terminals of τ into a single new terminal. Contraction of a triple is conveniently implemented by adding two new zero-cost edges $A(\tau) = \{e_1', e_2'\}$ between the three terminals of τ (see Figure 43.2[b]). It is easy to see that addition of $A(\tau)$ changes the MST of G_S—in the updated MST the two edges in $A(\tau)$ replace the two edges in $R(\tau)$. Finally, GTCA adds all chosen triples to the original MST of G_S and outputs the MST of this union.

Theorem 43.1 Berman et al. [27]

The cost of the rectilinear Steiner tree constructed by GTCA is at most 1.3125 times more than the optimal Steiner tree.

Fößmeier et al. [21] prove that to achieve an approximation ratio of 1.3125 in Theorem 43.1 it is sufficient to consider only *empty tree triples* of terminals. A triple τ is *empty* if the minimum rectangle bounding the triple does not contain any other terminals and is a *tree* triple if the center c of τ is adjacent to all terminals of the triple in $MST(S + c)$ (or, equivalently, if $gain(\tau) > 0$). As shown in Ref. [21], there are at most $36n$ empty tree triples. Even so, finding the best triple in step 3 of GTCA is very time consuming. An efficient

```
Input: Set $S$ of terminals
Output: 3-restricted Steiner tree $T$ spanning $S$

1. $T \leftarrow MST(G_S)$
2. $F \leftarrow \emptyset$
3. Repeat forever
       Find a triple $\tau$ with maximum gain
       If $gain\ (\tau) \leq 0$, then go to step 4
       $F \leftarrow F \cup \{\tau\}$                  // Add $\tau$
       $T \leftarrow T - R(\tau) + A(\tau)$           // Contract $\tau$
4. Output $MST(F \cup MST(G_S))$
```

FIGURE 43.3 The greedy triple contraction algorithm (From Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., *Proc. Asia and South Pacific Design Automation Conf.*, 2003, pp. 828–832.).

```
Input: Set $S$ of terminals
Output: Steiner tree $T$ spanning $S$

1. Compute the minimum spanning tree of $S$, $MST(S)$
2. Compute a set $Triples$, of size $O(n \log n)$, containing all empty tree triples
3. $SP \leftarrow \emptyset$
4. While $Triples \neq \emptyset$ do
      For each $\tau \in Triples$ compute $R(\tau)$, $A(\tau)$, and $gain(\tau)$, discarding triples with nonpositive gain
      Sort $Triples$ in descending order of gain
      Unmark all edges of $MST(S)$
      For each $\tau \in Triples$ do
         If both edges in $R(\tau)$ are unmarked, then mark them and replace them in the MST with the two
               edges in $A(\tau)$, i.e., $MST(S) \leftarrow MST(S) - R(\tau) + A(\tau)$
         $SP \leftarrow SP + center(\tau)$
5. If $SP = \emptyset$ then return the minimum spanning tree of $S$, else
      $S \leftarrow S + SP$
      Compute the minimum spanning tree of $S$ and discard all Steiner points with degree 1 or 2
      Go to step 2
```

FIGURE 43.4 The batched greedy algorithm (From Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., *Proc. Asia and South Pacific Design Automation Conf.*, 2003, pp. 828–832.).

$O(n^2 \log n)$ time implementation of GTCA should maintain dynamic MSTs for which, to date, there is no data structure able to handle instances with tens of thousands of nodes in practical running time. Existing data structures are difficult to implement and involve big asymptotic constants, see Cattaneo et al. [28] for a recent empirical study.

Our new heuristic, the BGA (see Figure 43.4) adopts the batched method from Ref. [17], substantially reducing running time by relaxing the greedy rule used to select triples in GTCA. After contracting a triple we continue by picking the best triple among those with unchanged gain; in general this may not be the best triple overall. Note that a triple τ can change its gain only if one of the edges in $R(\tau)$ is removed when contracting other triple—if none of the contracted triples removes edges from $R(\tau)$ then the gain of τ is unchanged. When done with one such *batched phase* (the body of the while loop in step 4) it is still possible to have positive gain triples. Therefore, we recompute triple gains and repeat the batched phase selection until no positive gain triples are left. To enable further improvements, we add the centers of triples selected in step 4 to the terminal set then iterate steps 2–5 (which constitute a *round* of the algorithm) until no more centers are added to the tree.

In next section we show how to compute in $O(n \log n)$ time a set of $O(n \log n)$ triples containing all empty tree triples (see Theorem 43.3). Then, in Section 43.4 we describe a data structure which enables computing a bottleneck edge on the tree path between any two given nodes in $O(\log n)$ time after $O(n \log n)$ time preprocessing. Since computing the gain of a triple amounts to three bottleneck edge computations, this leads to an $O(n \log^2 n)$ time implementation of the batched phase algorithm. This gives the following.

Theorem 43.2

The running time of the BGA is $O(Pn \log^2 n)$, where P is the total number of batched phases and n is the number of terminals.

In practice, the total number of phases P is small and can be bounded by a constant. Thus, the runtime of BGA is $O(n \log^2 n)$.

43.3 Generation of Triples

In this section we show how to compute in $O(n \log n)$ time a set of $O(n \log n)$ triples containing all empty tree triples. For simplicity, we assume that terminals are in general position, i.e., no two of them share the same x- or y-coordinate. This assumption is not restrictive since we can always break ties, for example, according to terminal IDs.

In a triple, the terminal which does not share x- and y-coordinates with the center (see the shaded portion of Figure 43.1) is called *diagonal.* There are four types of triples depending on where the diagonal terminal lies with respect to the center: a triple is called *north-west* if the diagonal terminal is in the northwest quadrant of the center (see Figure 43.1); north-east, south-west, and south-east triples are defined similarly. We will use the divide and conquer method to find $O(n \log n)$ northwest triples containing all northwest empty tree triples. Triples of the other types are obtained by reflection and application of the same algorithm.

For finding northwest triples we recursively partition the terminals into (almost) equal halves with a bisector line parallel to line $y = -x$. Let *LB* (left-bottom) and *TR* (top-right) be the half-planes defined by the bisector line, and let *D*, *R*, and *B* be the diagonal, right, and bottom terminals of a northwest triple that is intersected by the bisector line (see Figure 43.5). We distinguish the following four cases:

Case 1. D, R ∈ TR and B ∈ LB. Figure 43.6 (a) illustrates how to compute for each diagonal terminal *D* the unique terminal *R* that can serve as a right terminal in an empty northwest triple with *D* as the diagonal terminal. All terminals in *TR* are processed in x-ascending order as follows: (1) if the next terminal has y larger then the current terminal, then a dashed pointer is set from the next to the current terminal, and then the current terminal is advanced to the next terminal; (2) otherwise, a solid pointer is set from the current terminal to the next one, and the current terminal is moved back along the dashed pointer (if it

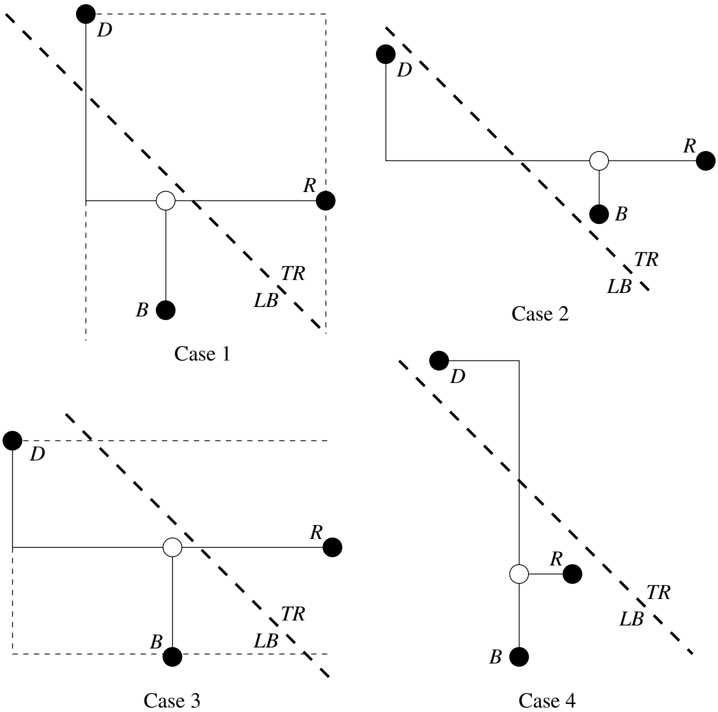


FIGURE 43.5 Four cases of partitioning of a northwest triple (From Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., *Proc. Asia and South Pacific Design Automation Conf.*, 2003, pp. 828–832.).

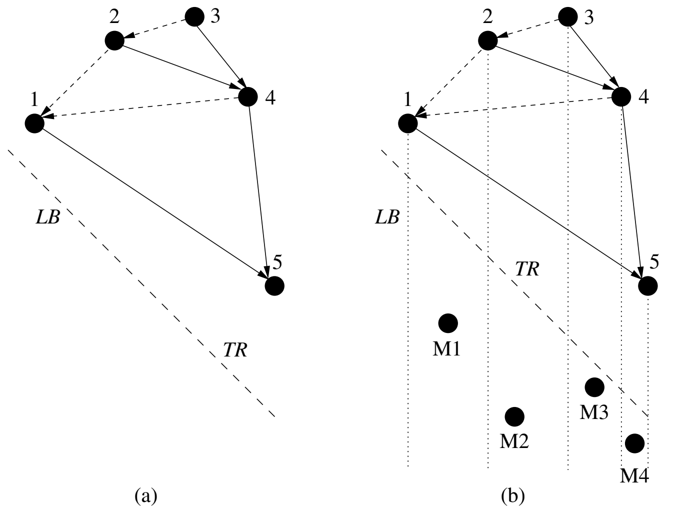


FIGURE 43.6 Case 1: (a) Finding the right terminal for each diagonal terminal, for example, if $D = 1$, then $R = 5$, if $D = 3$, then $R = 4$, etc. (b) Finding highest terminals in the strips corresponding to consecutive terminals (From Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., *Proc. Asia and South Pacific Design Automation Conf.*, 2003, pp. 828–832.).

exists, otherwise the current terminal is advanced to the next). Clearly, this procedure is linear since the runtime is proportional to the number of pointers established and each terminal has at most two pointers (one solid and one dashed). When processing of the points in *TR* is finished, each solid arc connects a terminal D with the leftmost terminal in TR lower than and to the right of D, that is, with the unique terminal R that can serve as a right terminal in an empty northwest triple with D as the diagonal terminal.

To find all case 1 northwest triples, we must find for each solid arc (D, R) in *TR* the node B in *LB* which can complete the triple, that is, the node B with maximum y-coordinate in the vertical strip defined by D and R. This is done in linear time by one traversal of the terminals in LB in x-ascending order (i.e., strip by strip) while computing the highest point in each strip.

Case 2. B, $R \in TR$ and $D \in LB$. For each terminal R, the unique terminal in *TR* that can serve as the bottom terminal in an empty northwest triple with R as the right terminal (i.e., the highest terminal in *TR* lower and to the left of R) can be found by a procedure similar to the one in step 1. Cf. [21], an arc (R, B) in *TR* is completed into a tree northwest triple only when the diagonal node D is the closest to R (and therefore to B) in the octant of *LB* containing points higher than R. To find the diagonal points D for each arc (R, B) in *TR*, we simultaneously traverse terminals in *TR* in y-ascending order and terminals in *LB* in $(x - y)$-ascending order as follows:
While there are unprocessed terminals in both *TR* and *LB*

- advance in *TR* until we reach a terminal R that has an arc to the associated B,
- advance in *LB* until we reach a terminal D higher than R,
- assign D to R.

Note that triples found by the above procedure are not necessarily empty. With a more careful implementation it is possible to avoid generating nonempty triples, however this would not change the asymptotic number of triples generated or the worst-case running time of the algorithm.

Case 3. $R \in TR$ and D, $B \in LB$. It is equivalent to case 1 after reflection over bisector.

Case 4. $D \in TR$ and B, $R \in LB$. It is equivalent to case 2 after reflection over bisector.

Theorem 43.3

A set of size $O(n \log n)$ containing all empty tree triples can be computed in $O(n \log n)$ time.

Proof

Each northwest empty tree triple crossing the dividing diagonal must fall in one of the four cases considered and finding all crossing triples takes linear time. Thus, the running time is given by the recurrence $T(n) = 2T(n/2) + O(n)$, that is, $T(n) = O(n \log n)$. The number of triples generated by the divide-and-conquer algorithm is also $O(n \log n)$ by the same recurrence; notice that each recursive step generates a linear number of triples. The same argument applies to the other three triple types. □

43.4 Computing Maximum Cost Edge on a Tree Path

It is easy to see that computing the gain of a triple τ and the edges in $R(\tau)$ reduces to finding bottleneck (i.e., most expensive) edges on the tree paths between pairs of terminals in τ. The *hierarchical greedy preprocessing* (HGP) algorithm given in Figure 43.7 computes for a given tree on n terminals two auxiliary arrays, *parent* and *edge*, with at most $2n - 1$ elements each. Using these arrays, the bottleneck tree edge between any two terminals u and v can be found in $O(\log n)$ using the algorithm in Figure 43.8.

Assuming that edges are sorted in ascending order of cost, HGP is equivalent to the following recursive procedure. First, for each node u, direct the cheapest edge incident to u, away from u, and save its index in *edge*(u). As a result some edges remain undirected, some become unidirected, and some become bidirected. In the subgraph induced by the (uni- and bi-) directed edges, each connected component consists of a

```
Input: Weighted tree $T = (V, E, cost)$ with $V = \{1, 2, ..., n\}$
Output: Arrays $parent(i)$ and $edge(i)$, $i = 1, ..., 2n - 1$

1. Sort tree edges $e_1, ..., e_{n-1}$ in ascending order of cost
2. Initialization:
     $next \leftarrow n$
     For each $i = 1, 2, ..., 2n - 1$ do
         $parent(i) \leftarrow NIL$
         $edge(i) \leftarrow NIL$
3. For each edge $e_i = (u, v)$, $i = 1, ..., n - 1$, do
     While $u \neq v$ and $parent(u) \neq NIL$ and $parent(v) \neq NIL$ do
         $u \leftarrow parent(u)$
         $v \leftarrow parent(v)$
     If $parent(u) = parent(v) = NIL$, then
         $next \leftarrow next + 1$
         $parent(u) \leftarrow parent(v) \leftarrow next$
         $edge(u) \leftarrow edge(v) \leftarrow i$
     If $parent(u) = NIL$ and $parent(v) \neq NIL$, then
         $parent(u) \leftarrow parent(v)$
         $edge(u) \leftarrow i$
     If $parent(u) \neq NIL$ and $parent(v) = NIL$, then
         $parent(v) \leftarrow parent(u)$
         $edge(v) \leftarrow i$
4. Output the arrays $parent(i)$ and $edge(i)$
```

FIGURE 43.7 The hierarchical greedy preprocessing algorithm (From Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., *Proc. Asia and South Pacific Design Automation Conf.*, 2003, pp. 828–832.).

```
Input: Tree edges $e_1, ..., e_{n-1}$ in ascending order of cost, arrays $parent(i)$ and $edge(i)$, $i = 1, ..., 2n - 1$,
and nodes $u, v \in V$
Output: Maximum cost edge on the tree path between $u$ and $v$

1. $index \leftarrow -\infty$,
2. While $u \neq v$ do
       $index \leftarrow \max\{index, edge(u), edge(v)\}$
       $u \leftarrow parent(u)$
       $v \leftarrow parent(v)$
3. Return $e_{index}$
```

FIGURE 43.8 Subroutine for computing the maximum cost edge on the tree path between nodes u and v (From Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., *Proc. Asia and South Pacific Design Automation Conf.*, 2003, pp. 828–832.).

bidirected edge with two (possibly empty) arborescences attached to its ends. HGP collapses each such connected component K into a single node q, then sets *parent*(u) to q for every $u \in K$. Since each connected component contains at least one bidirected edge, no more than $n/2$ collapsed component nodes are created. The procedure is repeated on the tree induced by collapsed components until there is a single node left. The total runtime of HGP is $O(n \log n)$ because of the edge sorting in step 1, remaining HGP steps require $O(n)$ time.

Clearly, edge costs decrease along any directed path of a connected component K. Therefore, if u and v are two vertices of K, then the index of the maximum cost edge on the tree path between u and v is max{*edge*(u), *edge*(v)}. If u and v are in different components K and K', we need to compute the maximum between *edge*(u), *edge*(v), and the maximum index of the most expensive edge on the path between K and K' in the tree T with collapsed connected components. The algorithm in Figure 43.8 is an iterative implementation of this recursive definition. Since the hierarchy of collapsed connected components has a depth of at most log n, we get Theorem 43.4.

Theorem 43.4

The algorithm in Figure 43.8 finds the maximum cost edge on the tree path connecting two given nodes in $O(\log n)$ *time after* $O(n \log n)$ *time for HGP.*

43.5 Experimental Results

Comprehensive experimental evaluation indicates that the iterated 1-Steiner heuristic of Kahng and Robins [17] significantly outperforms in solution quality the rectilinear Steiner tree heuristics proposed prior to 1992 [29]. Since then, the edge-based heuristic of Borah et al. [20], and the IRV heuristic [18] have been reported to match or slightly exceed iterated 1-Steiner in solution quality. However, among these best-performing heuristics only the edge-based heuristic can be applied to instances with tens of thousands of terminals, since current implementations of iterated 1-Steiner and IRV require quadratic memory. Besides Borah's $O(n^2)$ implementation of the edge-based heuristic, we compared our $O(n \log^2 n)$ BGA to the recent $O(n \log^2 n)$ Prim-based heuristic of Rohe [30]. For comparison purposes, we also include results from our implementation of the Guibas–Stolfi $O(n \log n)$ rectilinear MST algorithm [31], and, whenever possible, the optimum Steiner trees computed using the beta version of the GeoSteiner 4.0 algorithm in Ref. [14].

All heuristics and MST algorithms were run on a dual 1.4 GHz Pentium III Xeon server with 2 GB of memory running Red Hat Linux 7.1. The GeoSteiner code using the CPLEX 6.6 linear programming solver was run on a 360 MHz SUN Ultra 60 workstation with 2 GB of memory under SunOS 5.7. The test bed for our experiments consisted of two categories of instances: instances drawn uniformly at random

TABLE 43.1 Percent Improvement over MST and CPU Time of the Compared Rectilinear Steiner Tree Algorithms

	MST		Prim-based		Edge-based		Batched greedy		GeoSteiner 4.0	
#Term.	Len.(μm)	CPU	%Imp.	CPU	%Imp.	CPU	%Imp.	CPU	%Imp.	CPU
Random instances (average results over 10 instances)										
100	85169.9	0.0005	9.78	0.001	10.97	0.006	10.99	0.003	11.66	0.555
500	184209.7	0.0036	10.08	0.007	11.12	0.216	11.17	0.081	11.76	15.205
1000	258926.8	0.0079	10.04	0.014	10.96	0.939	10.99	0.230	11.61	117.916
5000	573178.8	0.0501	10.02	0.082	11.02	56.348	11.05	1.903	—	—
10000	809343.5	0.1268	10.04	0.191	11.01	415.483	11.05	5.192	—	—
50000	1808302.7	1.2330	10.05	1.320	11.01	16943.777	11.06	69.043	—	—
100000	2555821.9	3.1150	10.08	3.143	11.04	61771.928	11.08	195.589	—	—
500000	5710906.8	22.9130	10.07	20.570	—	—	11.08	1706.765	—	—
VLSI instances										
337	247.7	0.0020	5.96	0.000	6.50	0.060	6.43	0.040	6.75	16.040
830	675.6	0.0055	3.10	0.010	3.19	0.320	3.20	0.080	3.26	9.480
1944	452.2	0.0165	6.86	0.040	7.77	3.640	7.85	0.400	8.15	1304.270
2437	578.8	0.0217	7.09	0.040	7.96	5.740	7.96	0.680	8.34	13425.310
2676	887.2	0.0235	8.07	0.040	8.99	5.340	8.93	0.770	9.38	430.800
12052	2652.7	0.1378	7.65	0.180	8.46	540.840	8.45	5.230	—	—
22373	13962.5	0.3419	8.99	0.480	9.83	2263.760	9.85	13.060	—	—
34728	9900.5	0.5455	8.16	0.690	9.01	5163.060	9.05	24.200	—	—

Source: From Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., *Proc. Asia and South Pacific Design Automation Conf.*, 2003, pp. 828–832.

from a 1, 000, 000 × 1, 000, 000 grid, ranging in size between 100 and 500,000 terminals, and a set of eight test cases extracted from recent industrial designs, ranging in size between 330 and 34,000 terminals.

Table 43.1 gives the percent improvement over the rectilinear MST and running time (in CPU seconds) for experiments on rectilinear instances. On random instances, the batched greedy heuristic matches or slightly exceeds in average solution quality the edge-based heuristic of Ref. [20]. Both batched greedy and the edge-based heuristic improve the rectilinear MST by an average of 11% in our experiments. This is roughly 1% more than the average improvement achieved by the Prim-based heuristic of Ref. [30], and is within 0.7% of the optimum average improvement for the sizes for which the optimum could be computed using GeoSteiner. Results on VLSI instances show that the relative performance of the heuristics is the same to that observed on random instances. However, the improvement over the rectilinear MST and the gaps between heuristics are smaller in this case.

The results in Table 43.1 show that the BGA is highly scalable. Even though batched greedy is not as fast as the MST or the Prim-based heuristic of Ref. [30], it can easily handle up to hundreds of thousands of terminals in minutes of CPU time. Compared to Borah's $O(n^2)$ implementation of the edge-based heuristic, batched greedy is two or more orders of magnitude faster as soon as the number of terminals gets into the tens of thousands.

The BGA can be easily adapted to other cost metrics. We have implemented and experimented with an octilinear version of the BGA. The only required modifications are in the distance formula and in the procedure for finding the optimum Steiner point of a triple. The octilinear distance between points (x, y) and (x', y') is equal to $\max\{|x - x'|, |y - y'|\} + (\sqrt{2} - 1)\min\{|x - x'|, |y - y'|\}$; this is always smaller than the rectilinear distance, $|x - x'| + |y - y'|$, unless the two points are on the same horizontal or vertical line, in which case the two distances are equal. The computation of the optimum Steiner point of a triple in the octilinear metric can be done in constant time using the method described in Ref. [3].

Table 43.2 gives results obtained by the octilinear versions of the Guibas-Stolfi MST, $O(n^2)$ edge-based, batched greedy, and GeoSteiner 4.0 [32] algorithms. Octilinear batched greedy is almost always better than the octilinear edge-based heuristic, and very close to optimum for the instances for which the latter is

TABLE 43.2 Percent Improvement over MST and CPU Time of the Compared Octilinear Steiner Tree Algorithms

	MST		Edge-based		Batched greedy		GeoSteiner 4.0	
#Term.	Len.(μm)	CPU	%Imp.	CPU	%Imp.	CPU	%Imp.	CPU
Random instances (average results over 10 instances)								
100	72375.1	0.0005	4.28	0.530	4.43	0.010	4.75	11.608
500	155611.7	0.0036	4.12	13.410	4.29	0.118	4.60	311.991
1000	219030.8	0.0079	4.12	54.641	4.25	0.296	4.59	1321.382
5000	484650.5	0.0506	4.17	1466.296	4.31	2.820	—	—
10000	684409.5	0.1217	4.13	5946.815	4.28	8.362	—	—
50000	1528687.2	1.1940	4.16	147210.395	4.30	116.419	—	—
100000	2160629.4	3.1060	—	—	4.32	476.307	—	—
500000	4826839.1	23.0610	—	—	4.31	6578.840	—	—
VLSI instances								
337	219.0	0.0020	2.92	5.690	2.99	0.050	3.13	72.960
830	630.4	0.0055	0.93	27.610	0.90	0.120	1.07	195.190
1944	407.2	0.0167	3.33	202.030	3.47	0.750	4.01	5279.870
2437	523.1	0.0218	3.67	345.330	3.77	0.820	4.29	7484.730
2676	780.2	0.0236	3.41	392.340	3.51	1.310	3.89	6080.050
12052	2372.3	0.1417	3.63	7517.680	3.72	10.800	—	—
22373	12069.8	0.3447	3.65	25410.340	3.74	21.380	—	—
34728	8724.9	0.5427	3.64	62971.090	3.74	25.160	—	—

Source: From Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., *Proc. Asia and South Pacific Design Automation Conf.*, 2003, pp. 828–832.

available. Furthermore, octilinear batched greedy remains highly scalable, with just a small factor increase in runtime compared to the rectilinear version.

43.6 Conclusions

Noncritical nets with tens of thousands of terminals are becoming more common in modern designs because of the increased emphasis on design for test. Even a single net of this size can render quadratic Steiner tree algorithms impractical, given the stringent constraints on routing runtime (e.g., designers expect full chip global and detailed routing to be completed overnight). In this chapter we have given a high-quality $O(n \log^2 n)$ heuristic that can practically handle these nets without compromising solution quality. Since our heuristic is graph-based, it can be easily modified to handle other practical considerations, such as routing obstacles, preferred directions, and via costs. A C++ implementation of BGA is freely available as part of the MARCO Gigascale Silicon Research Center VLSI CAD Bookshelf at http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/FastSteiner/.

Acknowledgments

A preliminary version of this work has appeared in Ref. [24]. The authors wish to thank Manjit Borah, Benny Nielsen, André Rohe, and Martin Zachariasen for giving us access to their implementations; Charles Alpert for providing the VLSI benchmarks; and Jarrod Alexander Roy and Igor Markov for correcting a bug in our original implementation of the BGA algorithm. This work has been supported in part by Cadence Design Systems, Inc., the MARCO Gigascale Silicon Research Center, NSF Grant CCR-9988331, Award No. MM2-3018 of the Moldovan Research and Development Association (MRDA) and the U. S. Civilian Research & Development Foundation for the Independent States of the Former Soviet Union (CRDF), and by the State of Georgia's Yamacraw Initiative.

References

[1] Chen, H., Cheng, C. K., Kahng, A. B., Măndoiu, I. I., and Wang, Q., Estimation of wirelength reduction for λ-geometry vs. Manhattan placement and routing, *Proc. ACM Int. Workshop on System-Level Interconnect Prediction (SLIP)*, 2003, p. 71.

[2] Chen, H., Cheng, C.-K., Kahng, A. B., Măndoiu, I. I., Wang, Q., and Yao, B., The Y-architecture for on-chip interconnect: analysis and methodology, *IEEE Trans. CAD,* 24, 588, 2005.

[3] Koh, C.-K. and Madden, P. H., Manhattan or Non-Manhattan? A study of alternative VLSI routing architectures, *Proc. Great Lakes Symp. on VLSI*, 2000, p. 47.

[4] Scepanovic, R., Koford, J. S., Kudryavstev, V. B., Andreev, A. E., Aleshin, S. V., and Podkolzin, A. S., Microelectronic Integrated Circuit Structure and Method Using Three Directional Interconnect Routing Based on Hexagonal Geometry, U. S. Patent, No. US5578840, November 1996.

[5] Teig, S., The X architecture: not your father's diagonal wiring, *Proc. ACM/IEEE Workshop on System Level Interconnect Prediction (SLIP)*, 2002, p. 33.

[6] Teig, S. and Ganley, J. L., Method and Apparatus for Considering Diagonal Wiring in Placement, International Application, No. WO 02/47165 A2, 2002.

[7] `http://www.xinitiative.org`

[8] Sarrafzadeh, M. and Wong, C. K., Hierarchical Steiner tree construction in uniform orientations, *IEEE Trans. CAD*, 11, 1095, 1992.

[9] Koh, C. K., Steiner Problem in Octilinear Routing Model, Master's thesis, National University of Singapore, 1995.

[10] Li, Y. Y., Cheung, S. K., Leung, K. S., and Wong, C. K., Steiner tree constructions in λ_3-metric, *IEEE Trans. Circuits Syst.-II: Analog and Digital Signal Process.*, 45, 563, 1998.

[11] Hanan, M., On Steiner's problem with rectilinear distance, *SIAM J. Appl. Math.,* 14, 255, 1966.

[12] Chiang, C., Su, Q., and Chiang, C.-S., Wirelength reduction by using diagonal wire, *Proc. Great Lakes Symp. on VLSI*, 2003, p. 104.

[13] Coulston, C., Constructing exact octagonal Steiner minimal trees, *Proc. Great Lakes Symp. on VLSI*, 2003, 1.

[14] Nielsen, B. K., Winter, P., and Zachariasen, M., An exact algorithm for the uniformly-oriented Steiner tree problem, *Proc. of ESA*, Lecture Notes in Computer Science, Vol. 2461, Springer, Berlin, 2002, p. 760.

[15] Scheffer, L., Rectilinear MST code, available as part of the RMST-Pack at http://www.gigascale.org/bookshelf/slots/RSMT/RMST/.

[16] Zhou, H., Shenoy, N., and Nicholls, W., Efficient minimum spanning tree construction without Delaunay triangulation, *Proc. Asia-Pacific Design Automation Conf. (ASP-DAC)*, 2001, p. 192.

[17] Kahng, A. B. and Robins, G., A new class of iterative Steiner tree heuristics with good performance, *IEEE Trans. CAD*, 11, 1462, 1992.

[18] Măndoiu, I. I., Vazirani, V. V., and Ganley, J. L., A new heuristic for rectilinear Steiner trees, *IEEE Trans. CAD*, 19, 1129, 2000.

[19] Yildiz, M. C. and Madden, P. H., Preferred direction Steiner trees, *Proc. Great Lakes Symp. on VLSI*, 2001, p. 56.

[20] Borah, M., Owens, R. M., and Irwin, M. J., A fast and simple Steiner routing heuristic, *Disc. Appl. Math.,* 90, 51, 1999.

[21] Fößmeier, U., Kaufmann, M., and Zelikovsky, A., Faster approximation algorithms for the rectilinear Steiner tree problem, *Discrete Comput. Geometry,* 18, 93, 1997.

[22] Zelikovsky, A., An 11/6-approximation for the network Steiner tree problem, *Algorithmica*, 9, 463, 1993.

[23] Zachariasen, M., Personal communication, August 2002.

[24] Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., Highly scalable algorithms for rectilinear and octilinear Steiner trees, *Proc. Asia and South Pacific Design Automation Conf.*, 2003, p. 827.

[25] Zhu, Q., Zhou, H., Jing, T., Hong, X.-L., and Yang, Y., Spanning graph based non-rectilinear Steiner tree algorithms, *IEEE Trans. CAD,* 24, 2005, 1066.

[26] Gilbert, E. N. and Pollak, H. O., Steiner minimal trees, *SIAM J. Appl. Math.,* 32, 826, 1977.

[27] Berman, P., Fossmeier, U., Kaufmann, M., Karpinski, M., and Zelikovsky, A., Approaching the 5/4-approximation for rectilinear Steiner trees, *Proc. of ESA*, Lecture Notes in Computer Science, 762, 1994, 533.

[28] Cattaneo, G., Faruolo, P., Petrillo, U. F., and Italiano, G. F., Maintaining dynamic minimum spanning trees: an experimental study, in *Proc. 4th Int. Workshop on Algorithm Engineering and Experiments (ALENEX)*, Lecture Notes in Computer Science, 2409, Springer, Berlin, 2002, p. 111.

[29] Hwang, F. K., Richards, D. S., and Winter, P., *The Steiner tree problem*, Annals of Disc. Math., Vol. 53, North-Holland, Amsterdam, 1992.

[30] Rohe, A., Sequential and Parallel Algorithms for Local Routing, Ph.D. thesis, Bonn University, Bonn, Germany, 2001.

[31] Guibas, L. J. and Stolfi, J., On computing all north-east nearest neighbors in the L_1 metric, *Inf. Process. Lett.*, 17, 219, 1983.

[32] Warme, D. M., Winter, P., and Zacharisen, M., GeoSteiner 3.1 package, available from `http://www.diku.dk/geosteiner/`

44

Approximation Algorithms for Imprecise Computation Tasks with 0/1 Constraint

Joseph Y.-T. Leung
New Jersey Institute of Technology

44.1 Introduction

Meeting deadline constraints is of great importance in real-time systems. Sometimes, it is not feasible to schedule all the tasks so that they can meet their deadlines, a situation that occurs quite often when the system is saturated. In situations like this, it is often more desirable to execute some parts of every task, than to give up completely the execution of some tasks. The Imprecise Computation Model was introduced [1–3] to allow for the trade-off of the quality of computations in favor of meeting the deadline constraints. In this model, a task is logically decomposed into two subtasks, mandatory and optional. The mandatory subtask of each task is required to be completed by its deadline, while the optional subtask can be left unfinished. If a task has an unfinished optional subtask, it incurs an error equal to the execution time of its unfinished portion. The Imprecise Computation Model is extremely attractive in modeling iterative algorithms, where a task can be terminated prematurely with some loss of accuracy.

In the Imprecise Computation Model, each task T_i is represented by the quadruple $T_i = (r_i, d_i, m_i, o_i)$, where r_i, d_i, m_i, and o_i denote its release time, deadline, mandatory subtask's execution time, and optional subtask's execution time, respectively. Let $e_i = m_i + o_i$ denote its total execution time. A schedule for a given task system is *feasible* if each mandatory subtask is fully executed in the time interval between its release time and its deadline; a task system is *feasible* if there is a feasible schedule for it. Feasibility of a task system can be determined in at most $O(n^2 \log^2 n)$ time for a multiprocessor system [4] and $O(n \log n)$ time for a single processor [5]. In this chapter we assume that all task systems are feasible and all task parameters are rational numbers. Furthermore, we will be concerned with preemptive scheduling only.

Let S be a feasible schedule for a task system TS with n tasks. For each task T_i, let $\alpha(T_i, S)$ denote the amount of time T_i is executed in S. The *error* of T_i, denoted by $\epsilon(T_i, S)$, is defined to be $e_i - \alpha(T_i, S)$. The *total error* of S, denoted by $\epsilon(S)$, is defined to be $\sum_{i=1}^{n} \epsilon(T_i, S)$. The *maximum error* of S, denoted by $\Upsilon(S)$, is defined to be $\max_{i=1}^{n}\{\epsilon(T_i, S)\}$. The minimum of total error of TS, denoted by $\epsilon(TS)$, is defined to be $\min\{\epsilon(S) : S \text{ is a feasible schedule for } TS\}$. The minimum of maximum error of TS, denoted by $\Upsilon(TS)$, is similarly defined. If the importance of the tasks is not identical, we can assign a weight w_i to each task T_i, and the resulting task system is called a *weighted* task system. For a weighted task system, the goal is to minimize the total $w-$ *weighted* error. Sometimes we are interested in dual-criteria scheduling problems.

Scheduling problems with dual-criteria objective functions arise in some applications. In this case, we assign an additional weight w'_i to each task T_i. For dual-criteria scheduling problems, we are interested in minimizing the total $w - weighted$ error, subject to the constraint that the maximum $w' - weighted$ error is minimum, or minimizing the maximum $w' - weighted$ error, subject to the constraint that the total $w - weighted$ error is minimum.

Blazewicz [6] was the first to study the problem of minimizing the total $w - weighted$ error for a special case of the weighted task system, where each task has its optional subtask only; that is, $m_i = 0$ for each $1 \leq i \leq n$. He showed that both parallel processor and uniform processor systems can be reduced to minimum-cost-maximum-flow problems which can be transformed to linear programming problems. Blazewicz and Finke [7] later gave faster algorithms for both problems; see also Leung [8]. Potts and Van Wassenhove [9] studied the same problem for a single processor, assuming that all tasks have identical release times and identical weights. They developed an $O(n \log n)$-time algorithm for preemptive scheduling and showed that the problem becomes NP-hard for nonpreemptive scheduling. Later on they introduced a polynomial approximation scheme and two fully polynomial approximation schemes for the nonpreemptive case [10].

For imprecise computation tasks, Shih et al. [4] gave an $O(n^2 \log^2 n)$-time algorithm to minimize the total error on a parallel processor system. Shih et al. [11] and Leung et al. [12] later presented a faster algorithm for a single processor that runs in $O(n \log n)$ time. For the weighted case, Shih et al. [4] again showed that the problem can be transformed to a minimum-cost-maximum-flow problem, thus giving an $O(n^2 \log^3 n)$-time algorithm for parallel processors. For a single processor, Shih et al. [11] gave a faster algorithm that runs in $O(n^2 \log n)$ time, which was subsequently improved by Leung et al. [12] to $O(n \log n + kn)$ time, where k is the number of distinct $w - weights$.

Ho et al. [13] gave an algorithm to minimize the maximum $w' - weighted$ error; see also the work of Ho [14]. The algorithm runs in $O(n^3 \log^2 n)$ time for parallel processors and $O(n^2)$ time for a single processor. They also studied dual-criteria scheduling problems. For the problem of minimizing the total $w - weighted$ error, subject to the constraint that the maximum $w' - weighted$ error is minimum, Ho et al. [13] gave an algorithm that runs in $O(n^3 \log^2 n)$ time for parallel processors and $O(n^2)$ time for a single processor. For the problem of minimizing the maximum $w' - weighted$ error, subject to the constraint that the total $w - weighted$ error is minimum, Ho and Leung [15] gave an algorithm that runs in $O(kn^3 \log^2 n)$ time for parallel processors and $O(kn^2)$ time for a single processor, where k is the number of distinct $w - weights$.

The Imprecise Computation Model has also been studied under the 0/1-constraint, where each optional subtask is either fully executed or totally discarded. With the 0/1-constraint, two problems have been studied: (1) minimize the total error and (2) minimize the number of *imprecisely scheduled* tasks (i.e., tasks whose optional subtasks have been discarded). The 0/1-constraint is motivated by some practical applications. In real life, many tasks can be implemented by either a fast or a slow algorithm, with the slow algorithm producing better quality results than the fast one. Due to deadline constraints, it may not be possible to meet the deadline of every task if each task were to execute the slow version. Thus, the problem of scheduling tasks with primary version (slow algorithm) and alternate version (fast algorithm) can be reduced to one of scheduling with 0/1-constraint. The execution time of the fast algorithm is the mandatory execution time, while the execution time of the slow algorithm is the total execution time.

Lawler [16] has given an algorithm to solve the problem $1 \mid pmtn, r_j \mid \sum w_j U_j$; that is, one processor preemptive scheduling to minimize the weighted number of tardy tasks where tasks have release dates. His algorithm runs in $O(nk^2 W^2)$ time, where n is the number of tasks, k is the number of release dates, and W is the total weights of the tasks. In the following we will show that Lawler's algorithm can be used to solve the above two 0/1-constraint scheduling problems.

Consider the problem of minimizing the total error. Let TS be a set of n imprecise computation tasks and let $\sigma = \sum_{i=1}^{n} o_i$. For each task T_i, we create two tasks, an M-task with execution time m_i and weight $\sigma + 1$ and an O-task with execution time o_i and weight o_i. Both tasks have release time r_i and deadline d_i. It is clear that a schedule for the M-tasks and O-tasks that minimizes the weighted number of tardy tasks is also a feasible schedule that minimizes the total error for TS. Using Lawler's algorithm, such a schedule can be found in $O(n^5\sigma^2)$ time. Hence it can be solved in pseudopolynomial time. We note that the problem of minimizing the total error is NP-hard in the ordinary sense [11].

Now consider the problem of minimizing the number of imprecisely scheduled tasks. Let TS be a set of n imprecise computation tasks. For each task T_i, we create two tasks—an M-task with execution time m_i and weight $n+1$ and an O-task with execution time o_i and weight 1. Both tasks have release time r_i and deadline d_i. It is clear that a schedule for the M-tasks and O-tasks that minimizes the weighted number of tardy tasks is also a feasible schedule that minimizes the number of imprecisely scheduled tasks for TS. Using Lawler's algorithm, such a schedule can be found in $O(n^7)$ time.

Since the running times of the above algorithms are unacceptably high, there is a need for fast approximation algorithms. Ho et al. [17] gave two approximation algorithms, both of which run in $O(n^2)$ time. The first one, the *Largest-Optional-Execution-Time-First* algorithm, is used to minimize the total error. It has been shown [17] that the total error produced by the algorithm is at most three times that of the optimal solution and the bound is tight. The second algorithm, the *Smallest-Optional-Execution-Time-First* algorithm, is used to minimize the number of imprecisely scheduled tasks. It was shown [17] that the number of imprecisely scheduled tasks produced by the algorithm is at most two times that of an optimal solution and the bound is tight.

In this chapter we will present these two approximation algorithms. We begin by presenting an algorithm to test whether or not a set of tasks is feasible (Section 44.2). In Section 44.3, we will present the Largest-Optional-Execution-Time-First algorithm and prove that it has a worst-case bound of 3. In Section 44.4, we will present the Smallest-Optional-Execution-Time-First algorithm and prove that it has a worst-case bound of 2. In Section 44.5, we will prove two assertions that were used in the proofs of the previous two sections. Our concluding remarks appear in the last section.

We now define notations that will be used throughout this chapter. Let TS be a set of n imprecise computation tasks. We use $PO(TS)$ to denote the set of tasks with positive optional execution time. A task T_i is *eligible* in a given interval $[t', t"]$ if $r_i \leq t' \leq t" \leq d_i$. We use $\hat{\epsilon}(TS)$ to denote the minimum total error of TS under the 0/1-constraint; note that $\epsilon(TS)$ denotes the minimum total error of TS without any constraint. If H is a scheduling algorithm, we let (1) $E_H(TS)$ denote the set of precisely scheduled tasks produced by H; that is, $E_H(TS) = \{T_i : o_i > 0$ and T_i is precisely scheduled by $H\}$ and (2) $\tilde{E}_H(TS)$ denote the set of imprecisely scheduled tasks produced by H; that is, $\tilde{E}_H(TS) = PO(TS) - E_H(TS)$.

44.2 Feasibility Test

In this section, we will give a fast algorithm to test if a set of independent tasks is feasible on one processor. Each task T_i has an execution time e_i, release time r_i, and deadline d_i. The Boolean Function Feasible will decide if the set of tasks is feasible on one processor. Let $0 = min_{i=1}^{n}\{r_i\} = u_0 < u_1 < \cdots < u_p = max_{i=1}^{n}\{d_i\}$ be the $p+1$ distinct integers obtained from the multiset $\{r_1, \ldots, r_n, d_1, \ldots, d_n\}$. These $p+1$ integers divide the time frame into p segments: $[u_0, u_1], [u_1, u_2], \cdots, [u_{p-1}, u_p]$. The algorithm assumes that the tasks are indexed in descending order of release times and schedules the tasks in that order. When a task is scheduled, it is assigned from the latest segment in which it can be scheduled to the earliest one. Below is a formal description of the algorithm.

Boolean Function Feasible (TS)
Input: A task system $TS = (\{r_i\}, \{d_i\}, \{e_i\})$.
Output: "True" if TS is feasible and "False" otherwise.

Method:

```
(1) For $i = 1, \ldots, p$ do: $l_i \leftarrow u_i - u_{i-1}$.
(2) For $i = 1, \ldots, n$ do:

    Find $a$ satisfying $u_a = d_i$ and $b$ satisfying $u_b = r_i$.
    For $j = a, a-1, \ldots, b+1$ do:
        $\delta \leftarrow min\{l_j, e_i\}$.
        $l_j \leftarrow l_j - \delta$, $e_i \leftarrow e_i - \delta$.
        If $e_i = 0$ then "break"
    If $e_i > 0$ then return "False"
(3) Return "True"
```

Let us examine the time complexity of the algorithm. The time it takes to sort the tasks in descending order of their release times as well as obtaining the set $\{u_0, u_1, \ldots, u_p\}$ is $O(n \log n)$. Step 1 of the algorithm takes linear time and a straightforward implementation of step 2 takes $O(n^2)$ time. Thus, it appears that the running time of the algorithm is $O(n^2)$. However, observe that whenever a segment is scheduled, either all the units of a task are scheduled or the segment is saturated, or both. Hence, at most $O(n)$ segments have positive values. Thus, if we can avoid scanning those segments that have zero values, then step 2 takes only linear time. As it turns out, this can be done by the special UNION-FIND algorithm due to Gabow and Tarjan [18].

As we will see later, both of our approximation algorithms make n calls to Function Feasible, with the same set of tasks but different values of execution times. Since each call takes linear time, the overall running time of our approximation algorithm is $O(n^2)$.

44.3 Total Error

In this section we will give a fast approximation algorithm to minimize total error. The algorithm works as follows. Let *TS* be a set of n tasks and let $n' =\mid PO(TS) \mid$. First, the tasks are sorted in descending order of their optional execution times. Then, the set of precisely scheduled tasks is initialized to be the empty set and the following process is iterated n' times. At the ith iteration, we set the execution times of T_i and all the precisely scheduled tasks to be its mandatory plus optional execution times, and the execution times of all other tasks to be its mandatory execution times only. We then test if this set of tasks is feasible. If it is feasible, we include T_i into the set of precisely scheduled tasks; otherwise, T_i will not be included. A formal description of the algorithm is given below.

Algorithm Largest-Optional-Processing-Time-First
Input: A feasible task system $TS = (\{r_i\}, \{d_i\}, \{m_i\}, \{o_i\})$ consisting of n tasks with $n' = \mid PO(TS) \mid$.
Output: A feasible schedule S_L satisfying the 0/1-constraint.

Method:

```
(1) Sort the tasks in descending order of their optional execution times; that is, $o_i \geq o_{i+1}$ for $1 \leq i < n$.
    $E_L(TS) \leftarrow \emptyset$.
(2) For $i = 1, 2, \ldots, n'$ do:
        Create a set of $n$ tasks $TS'$ with release times, deadlines, and execution times as follows: For
        each task $T_j$ in $E_L(TS) \cup \{T_i\}$, create a task $T'_j$ with the same release time and deadline as $T_j$,
        and execution time $e'_j = m_j + o_j$. For each remaining task $T_k$, create a task $T'_k$ with the same
        release time and deadline as $T_k$, and execution time $e'_k = m_k$.
        If Feasible(TS') = "True," then $E_L(TS) \leftarrow E_L(TS) \cup \{T_i\}$.
```

It is clear that S_L is a feasible schedule satisfying the 0/1-constraint. The time complexity of the algorithm is $O(n^2)$. It is interesting to observe that the worst-case performance of the algorithm is unbounded if the tasks were sorted in ascending order, rather than descending order, of their optional execution times. Consider the two tasks T_1 and T_2 with $r_1 = 0$, $d_1 = x$, $m_1 = 0$, $o_1 = x$, $r_2 = 0$, $d_2 = 1$, $m_2 = 0$, and $o_2 = 1$. With the new ordering, T_2 will be scheduled precisely. However, the optimal solution will schedule T_1 precisely. Thus, the ratio becomes x which can be made arbitrarily large. However, if the tasks were sorted in descending order of their optional execution times, the algorithm has a worst-case bound at most 3, as the following theorem shows.

Theorem 44.1

For any task system TS, we have $\epsilon_L(TS) \leq 3\hat{\epsilon}(TS)$, where $\epsilon_L(TS)$ is the total error produced by Algorithm Largest-Optional-Execution-Time-First. Moreover, the bound can be achieved asymptotically.

We first give a task system showing that the bound can be achieved asymptotically. Consider four tasks T_1, T_2, T_3, and T_4 with $r_1 = x - \delta$, $d_1 = 2x - \delta$, $r_2 = 0$, $d_2 = x$, $r_3 = 2x - 2\delta$, $d_3 = 3x - 2\delta$, $r_4 = x$, $d_4 = 2x - 2\delta$, $m_1 = m_2 = m_3 = m_4 = 0$, $o_1 = o_2 = o_3 = x$, and $o_4 = x - 2\delta$. It is clear that Algorithm Largest-Optional-Execution-Time-First will schedule T_1 precisely while the optimal solution will schedule T_2, T_3, and T_4 precisely. Thus, the ratio approaches 3 as δ approaches 0.

We will prove Theorem 44.1 in the remainder of this section. First, we will state an assertion that will be proved in Section 44.5. Call a task *improper* if it is precisely scheduled by Algorithm Largest-Optional-Execution-Time-First, but not by the optimal algorithm.

Assertion 44.1

Let T_s be an improper task in the task system TS and let $\tilde{TS}$ be obtained from TS by setting the optional execution time of T_s to be 0. Then we have $\epsilon_L(\tilde{TS}) > \epsilon_L(TS) - 3o_s$.

With Assertion 44.1, we will prove the bound by contradiction. Let *TS* be the smallest task system, in terms of n', that violates the bound. We will characterize the nature of *TS* in the next two lemmas.

Lemma 44.1

$E_L(TS) \cap E_O(TS) = \emptyset$ and $E_L(TS) \cup E_O(TS) = PO(TS)$.

Proof

If $E_L(TS) \cap E_O(TS) \neq \emptyset$, let $T_i \in E_L(TS) \cap E_O(TS)$. Consider the task system $\tilde{TS}$ obtained from *TS* by setting the mandatory and optional execution times of T_i to be $m_i + o_i$ and 0, respectively. It is clear that $| PO(\tilde{TS}) | < | PO(TS) |$, $\epsilon_L(\tilde{TS}) = \epsilon_L(TS)$ and $\hat{\epsilon}(\tilde{TS}) = \hat{\epsilon}(TS)$. Thus, $\tilde{TS}$ is a smaller task system violating the bound, contradicting the assumption that *TS* is the smallest.

If $E_L(TS) \cup E_O(TS) \neq PO(TS)$, let T_i be a task such that $o_i > 0$ and $T_i \notin E_L(TS) \cup E_O(TS)$. Consider the task system $\tilde{TS}$ obtained from *TS* by setting the optional execution time of T_i to be 0. Clearly, $| PO(\tilde{TS}) | < | PO(TS) |$, $\epsilon_L(\tilde{TS}) = \epsilon_L(TS) - o_i$ and $\hat{\epsilon}(\tilde{TS}) = \hat{\epsilon}(TS) - o_i$. Thus, $\epsilon_L(\tilde{TS}) = \epsilon_L(TS) - o_i > 3\hat{\epsilon}(TS) - o_i = 3\hat{\epsilon}(\tilde{TS}) + 2o_i > 3\hat{\epsilon}(\tilde{TS})$, and hence $\tilde{TS}$ is a smaller task system violating the bound. □

Lemma 44.2

$E_L(TS) = \emptyset$.

Proof

If $E_L(TS) \neq \emptyset$, let $E_L(TS) = \{T_{i_1}, \ldots, T_{i_m}\}$, where $m \geq 1$. By Lemma 44.1, each task in $E_L(TS)$ is an improper task. Consider the task system $\tilde{TS}$ obtained from *TS* by setting the optional execution time of T_{i_m} to be 0. Clearly, $| PO(\tilde{TS}) | < | PO(TS) |$. By Assertion 44.1, we have

$$\epsilon_L(\tilde{TS}) > \epsilon_L(TS) - 3o_{i_m} \tag{44.1}$$

Since $\epsilon_L(TS) > 3\hat{\epsilon}(TS)$, we have

$$\epsilon_L(\tilde{TS}) > 3(\hat{\epsilon}(TS) - o_{i_m}) \tag{44.2}$$

Since $T_{i_m} \notin E_O(TS)$, we have

$$\hat{\epsilon}(\tilde{TS}) = \hat{\epsilon}(TS) - o_{i_m} \tag{44.3}$$

From Eq. (44.2) and Eq. (44.3), we have $\epsilon_L(\tilde{TS}) > 3\hat{\epsilon}(\tilde{TS})$, contradicting the assumption that *TS* is the smallest task system violating the bound. □

We can now prove Theorem 44.1. Lemma 44.2 implies that Algorithm Largest-Optional-Execution-Time-First cannot schedule *any* tasks precisely. However, Lemma 44.1 implies that an optimal algorithm schedules *every* task precisely. These two facts lead to impossibility.

44.4 Number of Imprecisely Scheduled Tasks

In this section we will give a fast approximation algorithm for minimizing the number of imprecisely scheduled tasks. The algorithm, to be called Algorithm Smallest-Optional-Execution-Time-First, works exactly like Algorithm Largest-Optional-Execution-Time-First, except that tasks are sorted in ascending order of their optional execution times. A formal description of the algorithm is given below.

Algorithm Smallest-Optional-Processing-Time-First
Input: A feasible task system $TS = (\{r_i\}, \{d_i\}, \{m_i\}, \{o_i\})$ consisting of n tasks with $n' =| PO(TS) |$.
Output: A feasible schedule S_S satisfying the 0/1-constraint.

Method:

```
(1) Index the tasks in $PO(TS)$ from 1 to $n'$ such that $o_i \le o_{i+1}$ for $1 \le i < n'$. Index the remaining
    tasks from $n' + 1$ to $n$. $E_S(TS) \leftarrow \emptyset$.
(2) For $i = 1, 2, \ldots, n'$ do:

        Create a set of $n$ tasks $TS'$ with release times, deadlines, and execution times as follows: For
        each task $T_j$ in $E_S(TS) \cup \{T_i\}$, create a task $T'_j$ with the same release time and deadline as $T_j$,
        and execution time $e'_j = m_j + o_j$. For every other task $T_k$, create a task $T'_k$ with the same
        release time and deadline as $T_k$, and execution time $e'_k = m_k$.

        If Feasible(TS$'$) = "True," then $E_S(TS) \leftarrow E_S(TS) \cup \{T_i\}$.
```

It is clear that S_S is a feasible schedule satisfying the 0/1-constraint. The time complexity of Algorithm Smallest-Optional-Execution-Time-First is $O(n^2)$. In the last section we showed that if Algorithm Smallest-Optional-Execution-Time-First were used to minimize total error, then it would give an unbounded performance. As it turns out, if Algorithm Largest-Optional-Execution-Time-First were used to minimize the number of imprecisely scheduled tasks, then it would also give an unbounded performance. Consider the task system consisting of $n + 1$ tasks: For $1 \le i \le n$, $r_i = (i - 1)x$, $d_i = ix$, $m_i = 0$, and $o_i = x$; $r_{n+1} = 0$, $d_{n+1} = nx$, $m_{n+1} = 0$, $o_{n+1} = nx$, where $x > 0$. It is clear that Algorithm Largest-Optional-Execution-Time-First schedules T_{n+1} precisely, while the optimal algorithm schedules $T_1, T_2, \ldots, T_n$ precisely. Thus, the ratio can be made arbitrarily large by taking n large enough. However, Algorithm Smallest-Optional-Execution-Time-First gives a much better performance, as the next theorem shows.

Theorem 44.2

For any task system TS, we have $| \tilde{E}_S(TS) \le 2 \mid \tilde{E}_O(TS) |$. *Moreover, the bound is tight.*

We first give a task system showing that the bound is tight. Consider the three tasks T_1, T_2, and T_3 with $r_1 = x - \delta$, $d_1 = 2x - \delta$, $r_2 = 0$, $d_2 = x$, $r_3 = 2(x - \delta)$, $d_3 = 3x - 2\delta$, $m_1 = m_2 = m_3 = 0$, $o_1 = o_2 = o_3 = x$, where $x > \delta$. Algorithm Smallest-Optional-Execution-Time-First schedules T_1 precisely, while the optimal algorithm schedules T_2 and T_3 precisely. Thus, the ratio is 2. In the following we will prove the upper bound. First, we will state an assertion that will be proved in the next section. Call a task *improper* if it is precisely scheduled by Algorithm Smallest-Optional-Execution-Time-First, but not by the optimal algorithm.

Assertion 44.2

Let T_s be an improper task in the task system TS and let $\tilde{TS}$ be obtained from TS by setting the optional execution time of T_s to be 0. *Then we have* $| \tilde{E}_S(\tilde{TS}) |\ge| \tilde{E}_S(TS) | -2$.

With Assertion 44.2, we will prove the upper bound by contradiction. Let *TS* be the smallest task system, in terms of n', that violates the bound. The next two lemmas, which are counterparts of Lemmas 44.1 and 44.2, characterize the nature of *TS*.

Lemma 44.3

$E_S(TS) \cap E_O(TS) = \emptyset$ *and* $E_S(TS) \cup E_O(TS) = PO(TS)$.

Proof

If $E_S(TS) \cap E_O(TS) \neq \emptyset$, let $T_i \in E_S(TS) \cap E_O(TS)$. Consider the task system $\tilde{TS}$ obtained from *TS* by setting the mandatory and optional execution times of T_i to be $m_i + o_i$ and 0, respectively. It is clear that $\mid PO(\tilde{TS}) \mid < \mid PO(TS) \mid$, $\tilde{E}_S(\tilde{TS}) = \tilde{E}_S(TS)$ and $\tilde{E}_O(\tilde{TS}) = \tilde{E}_O(TS)$. Thus, $\tilde{TS}$ is a smaller task system violating the bound, contradicting the assumption that *TS* is the smallest.

If $E_S(TS) \cup E_O(TS) \neq PO(TS)$, let T_i be a task such that $o_i > 0$ and $T_i \notin E_S(TS) \cup E_O(TS)$. Consider the task system $\tilde{TS}$ obtained from *TS* by setting the optional execution time of T_i to be 0. Clearly, $\mid PO(\tilde{TS}) \mid < \mid PO(TS) \mid$, $\mid \tilde{E}_S(\tilde{TS}) \mid = \mid \tilde{E}_S(TS) \mid -1$ and $\mid \tilde{E}_O(\tilde{TS}) \mid = \mid \tilde{E}_O(TS) \mid -1$. Thus, $\mid \tilde{E}_S(\tilde{TS}) \mid = \mid \tilde{E}_S(TS) \mid -1 > 2 \mid \tilde{E}_O(TS) \mid -1 = 2 \mid \tilde{E}_O(\tilde{TS}) \mid +1 > 2 \mid \tilde{E}_O(\tilde{TS}) \mid$, and hence $\tilde{TS}$ is a smaller task system violating the bound. □

Lemma 44.4

$E_S(TS) = \emptyset$.

Proof

If $E_S(TS) \neq \emptyset$, let $E_S(TS) = \{T_{i_1}, \ldots, T_{i_m}\}$, where $m \geq 1$. By Lemma 44.3, each task in $E_S(TS)$ is an inappropriate task in *TS*. Consider the task system $\tilde{TS}$ obtained from *TS* by setting the optional execution time of T_{i_m} to be 0. Clearly, $\mid PO(\tilde{TS}) \mid < \mid PO(TS) \mid$. By Assertion 44.2, we have

$$\mid \tilde{E}_S(\tilde{TS}) \mid \geq \mid \tilde{E}_S(TS) \mid -2 \tag{44.4}$$

Since $\mid \tilde{E}_S(TS) \mid > 2 \mid \tilde{E}_O(TS) \mid$, we have

$$\mid \tilde{E}_S(\tilde{TS}) \mid > 2 \mid \tilde{E}_O(TS) \mid -2 \tag{44.5}$$

Since $\mid \tilde{E}_O(\tilde{TS}) \mid = \mid \tilde{E}_O(TS) \mid -1$, we have

$$\mid \tilde{E}_S(\tilde{TS}) \mid > 2 \mid \tilde{E}_O(\tilde{TS}) \mid \tag{44.6}$$

contradicting our assumption that *TS* is the smallest task system violating the bound. □

Lemma 44.4 implies that Algorithm Smallest-Optional-Execution-Time-First cannot schedule *any* tasks in *PO*(*TS*) precisely, while Lemma 44.3 implies that the optimal algorithm schedules *every* task in *PO*(*TS*) precisely. These two facts lead to an impossibility, which proves Theorem 44.2.

While Theorem 44.2 gives a relationship between $\mid \tilde{E}_S(TS) \mid$ and $\mid \tilde{E}_O(TS) \mid$, it does not give a meaningful relationship between $\mid E_S(TS) \mid$ and $\mid E_O(TS) \mid$. The next theorem shows that they are also related by the same multiplicative factor.

Theorem 44.3

For any task system TS, $\mid E_O(TS) \mid \leq 2 \mid E_S(TS) \mid$. *Moreover, the bound is tight.*

Proof

The task system *TS* given in the proof of Theorem 44.2 also shows that $\mid E_O(TS) \mid = 2 \mid E_S(TS) \mid$. The upper bound is proved by contradiction. Let *TS* be the smallest task system, in terms of $\mid PO(TS) \mid$, that violates the bound. It is easy to verify that Lemma 44.3 also holds for *TS*. Thus, $E_S(TS) = \tilde{E}_O(TS)$ and $E_O(TS) = \tilde{E}_S(TS)$. Hence, $\mid E_O(TS) \mid = \mid \tilde{E}_S(TS) \mid \leq 2 \mid \tilde{E}_O(TS) \mid = 2 \mid E_S(TS) \mid$, contradicting our assumption that *TS* violates the bound. □

44.5 Proofs of Assertions

In this section we will prove Assertions 44.1 (in Section 44.3) and 44.2 (in Section 44.4), thereby completing all the proofs in this chapter. For convenience, we will state the assertions in the following.

Assertion 44.1

Let T_s be an improper task (with respect to Algorithm Largest-Optional-Execution-Time-First) in the task system TS and let $\tilde{TS}$ be obtained from TS by setting the optional execution time of T_s to be 0. *Then we have* $\epsilon_L(\tilde{TS}) > \epsilon_L(TS) - 3o_s$.

Assertion 44.2

Let T_s be an improper task (with respect to Algorithm Smallest-Optional-Execution-Time-First) in the task system TS and let $\tilde{TS}$ be obtained from TS by setting the optional execution time of T_s to be 0. *Then we have* $\mid \tilde{E}_S(\tilde{TS}) \mid \geq \mid \tilde{E}_S(TS) \mid -2$.

Let $\Gamma_L(T_s)$ be the set of all subsets of tasks in $PO(TS) - E_L(TS)$ such that it is feasible to schedule all the tasks in the subset precisely, along with all the tasks in $E_L(TS) - \{T_s\}$; that is, $\Gamma_L(T_s) = \{\tau_L : \tau_L \subseteq PO(TS) - E_L(TS)$ and there is a feasible schedule for TS such that all the tasks in $\tau_L \cup (E_L(TS) - \{T_s\})$ are precisely scheduled $\}$. Similarly, let $\Gamma_S(T_s)$ be the set of all subsets of tasks in $PO(TS) - E_S(TS)$ such that it is feasible to schedule all the tasks in the subset precisely, along with all the tasks in $E_S(TS) - \{T_s\}$; that is, $\Gamma_S(T_s) = \{\tau_S : \tau_S \subseteq PO(TS) - E_S(TS)$ and there is a feasible schedule for TS such that all the tasks in $\tau_S \cup (E_S(TS) - \{T_s\})$ are precisely scheduled $\}$. Then it is easy to see that Assertions 44.1 and 44.2 follow directly from the following assertions, 44.3 and 44.4, respectively.

Assertion 44.3

For any $\tau_L \in \Gamma_L(T_s)$, *we have* $\sum_{T_i \in \tau_L} o_i < 3o_s$.

Assertion 44.4

For any $\tau_S \in \Gamma_S(T_s)$, *we have* $\mid \tau_S \mid \leq 2$.

In the following we will prove Assertions 44.3 and 44.4. First, we need to prove two properties about $\Gamma_L(T_s)$ and $\Gamma_S(T_s)$. Since these properties are common to both, we will use H to denote both L and S. The two properties are:

Property 44.1

For any $\tau_H \in \Gamma_H(T_s)$, *we have* $\sum_{T_i \in \tau_H} o_i \leq \Delta_H(T_s) + o_s$, *where* $\Delta_H(T_s)$ *is the total amount of processor times that could possibly be assigned to the optional execution times of the tasks in* τ_H without *the* 0/1-*constraint, under the condition that all the tasks in* $E_H(TS)$ *are precisely scheduled.*

Property 44.2

$\Delta_H(T_s) < o_p + o_q$, *where* T_p *and* T_q *are the two tasks in* τ_H *with the earliest release time and the latest deadlines, respectively.*

To facilitate our proof, we will assume that an optimal schedule can be divided from time $\min\{r_i\}$ to time $\max\{d_i\}$ into equal-length segments such that there is no task reassignment within each segment; we use μ to denote the length of each segment. Such an assumption can be made because the task parameters are rational numbers and a feasible schedule can be constructed with only a finite number of preemptions. The following lemma, whose proof we will omit, is instrumental in proving Property 44.1.

Lemma 44.5

Let $\tilde{TS}$ be obtained from TS by reducing the mandatory execution time of T_i by an amount x, where $0 \leq x \leq m_i$. Then we have $\epsilon(TS) \leq \epsilon(\tilde{TS}) + x$.

Using the above lemma, we can show Property 44.1.

Lemma 44.6

For any $\tau_H \in \Gamma_H(T_s)$, we have $\sum_{T_i \in \tau_H} o_i \leq \Delta_H(T_s) + o_s$, where $\Delta_H(T_s)$ is the total amount of processor times that could possibly be assigned to the optional subtasks of the tasks in τ_H without the $0/1$-constraint, under the condition that all the tasks in $E_H(TS)$ are precisely scheduled.

Proof

Consider the two task systems $\tilde{TS}$ and $\hat{TS}$ obtained from TS as follows: In $\tilde{TS}$, the mandatory execution time of T_i is set to be $m_i + o_i$ if $T_i \in E_H(TS)$; otherwise, it is set to be m_i. The optional execution time of T_i is set to be o_i if $T_i \in \tau_H$; otherwise, it is set to be 0. The task system $\hat{TS}$ is the same as $\tilde{TS}$, except that the mandatory execution time of T_s is set to be m_s. Clearly, $\hat{TS}$ is obtained from $\tilde{TS}$ by reducing the mandatory execution time of T_s by o_s. Thus, by Lemma 44.5, we have

$$\epsilon(\tilde{TS}) \leq \epsilon(\hat{TS}) + o_s \tag{44.7}$$

Since it is feasible to schedule all the tasks in $\tau_H \cup (E_H(TS) - \{T_s\})$ precisely, we have $\epsilon(\hat{TS}) = 0$. Furthermore, it is easy to see that $\epsilon(\tilde{TS}) = \sum_{T_i \in \tau_H} o_i - \Delta_H(T_s)$. Substituting into 44.7, we obtain the desired result immediately. □

We now proceed to show Property 44.2. The next lemma is instrumental in proving this result.

Lemma 44.7

None of the tasks in τ_H can be precisely scheduled in any feasible schedule in which every task in $E_H(TS)$ is precisely scheduled.

Proof

If there were a task T_i in τ_H that can be precisely scheduled along with all the tasks in $E_H(TS)$, then T_i would be included in $E_H(TS)$, contradicting the fact that it is in τ_H. □

Consider the task system TS^* obtained from TS as follows: TS^* consists of all the tasks in TS plus an extra task T_{n+1} not in TS. In TS^*, the mandatory execution time of T_i, $1 \leq i \leq n$, is set to be $m_i + o_i$ if $T_i \in E_H(TS) - \{T_s\}$; otherwise, it is set to be m_i. The optional execution time of T_i, $1 \leq i \leq n$, is set to be o_i if $T_i \in \tau_H$; otherwise, it is set to be 0. Finally, the release time, deadline, mandatory, and optional execution times of T_{n+1} are set to be r_s, d_s, o_s, and 0, respectively.

Let S^* be the schedule such that $\epsilon(S^*) = \epsilon(TS^*)$. Divide S^* into equal-length segments and index them as $1, 2, \ldots, \lceil D/\mu \rceil$, where $D = max_{T_i \in TS^*}\{d_i\} - min_{T_i \in TS^*}\{r_i\}$ and μ is the length of the segments of S^*. Now remove the optional subtasks of all the tasks in τ_H from S^*. We will call the newly created idle segments the $\tau_H - idle$ segments. Since T_{n+1} acts like the optional subtask of T_s, $\Delta_H(T_s)$ is equal to the total length of all the $\tau_H - idle$ segments in S^*. Thus, our problem is reduced to finding an upper bound for the total length of all the $\tau_H - idle$ segments.

Let T_p be the task in τ_H such that $r_p = min_{T_i \in \tau_H}\{r_i\}$ and T_q be the task in τ_H such that $d_q = max_{T_i \in \tau_H}\{d_i\}$. By left or right-shifting tasks in S^* if necessary, we may assume that the execution of T_{n+1} in the interval $[r_p, d_q]$ is not separated by any $\tau_H - idle$ segment. It can be shown that all the $\tau_H - idle$ segments can be directly or indirectly used to schedule the optional subtask of T_p or T_q. Since neither T_p nor T_q can be precisely scheduled along with the tasks in $E_H(TS)$, the total length of all the $\tau_H - idle$ segments must be less than $o_p + o_q$.

The following two lemmas, whose proofs we will omit, bound the total length of all the $\tau_H - idle$ segments; see Ref. [17] for proofs.

Lemma 44.8

If T_{n+1} is executed in $[r_p, d_q]$ in S^*, then the total length of all the $\tau_H -$ idle segments is less than $o_p + o_q$.

Lemma 44.9

If T_{n+1} is not executed in $[r_p, d_q]$ in S^*, then the total length of all the $\tau_H -$ idle segments is less than o_p or o_q.

Lemmas 44.8 and 44.9 immediately imply Property 44.2.

Lemma 44.10

$\Delta_H(T_s) < o_p + o_q$, where T_p and T_q are the two tasks in τ_H with the earliest release time and the latest deadline, respectively.

We are now in a position to prove Assertions 44.3 and 44.4. First, we need to prove the next lemma.

Lemma 44.11

For each task $T_i \in \tau_L$, we have $o_i \le o_s$. For each task $T_j \in \tau_S$, we have $o_j \ge o_s$.

Proof

We prove the lemma by contradiction. Suppose that T_i is in τ_L and $o_i > o_s$. Since Algorithm Largest-Optional-Execution-Time-First considers the tasks in descending order of their optional execution times, T_i must be considered before T_s. Since T_i is in τ_L, it is feasible to schedule all the tasks in $\tau_L \cup (E_L(TS) - \{T_s\})$ precisely. But this means that T_i would be in $E_L(TS)$, contradicting the fact that it is in τ_L. The second part of the lemma can be proved in the same manner by observing that Algorithm Smallest-Optional-Execution-Time-First considers the tasks in ascending order of their optional execution times. □

Lemma 44.12

For any $\tau_L \in \Gamma_L(T_s)$, we have $\sum_{T_i \in \tau_L} o_i < 3o_s$.

Proof

By Lemmas 44.6 and 44.10, for any $\tau_L \in \Gamma_L(T_s)$, $\sum_{T_i \in \tau_L} o_i < o_p + o_q + o_s$, where T_p and T_q are the two tasks in τ_L with the earliest release time and the latest deadline, respectively. The lemma follows immediately from Lemma 44.11. □

Lemma 44.13

For any $\tau_S \in \Gamma_S(T_s)$, we have $\mid \tau_S \mid \le 2$.

Proof

By Lemmas 44.6 and 44.10, for any $\tau_S \in \Gamma_S(T_s)$, $\sum_{T_i \in \tau_S} o_i < o_p + o_q + o_s$, where T_p and T_q are the two tasks in τ_S with the earliest release time and the latest deadline, respectively. By Lemma 44.11, the optional execution time of every task in τ_S is at least o_s. Thus, $\mid \tau_S \mid \le 2$. □

44.6 Conclusions

In this chapter we have considered the problem of preemptively scheduling a set of imprecise computation tasks on a single processor, with the added constraint that each optional subtask is either fully executed or not executed at all. We gave an $O(n^2)$-time approximation algorithm, Algorithm Largest-Optional-Execution-Time-First, for minimizing the total error, and showed that it has a tight bound of 3. We also gave an $O(n^2)$-time approximation algorithm, Algorithm Smallest-Optional-Execution-Time-First, for minimizing the number of imprecisely scheduled tasks, and showed that it has a tight bound of 2. Interestingly, the number of precisely scheduled tasks in an optimal schedule is also bounded above by two times the number of precisely scheduled tasks in Algorithm Smallest-Optional-Execution-Time-First.

For future research, it will be interesting to see if there are any fast approximation algorithms for the two problems with better performance bounds than those of Algorithm Largest-Optional-Execution-Time-First and Algorithm Smallest-Optional-Execution-Time-First.

References

[1] Lin, K.-J., Natarajan, S., and Liu, J. W. S., Concord: a distributed system making use of imprecise results, *Proc. COMPSAC '87*, Tokyo, Japan, 1987.

[2] Lin, K.-J., Natarajan, S., and Liu, J. W. S., Imprecise results: utilizing partial computations in real-time systems, *Proc. 8th Real-Time Systems Symp.*, San Francisco, CA, 1987.

[3] Lin, K.-J., Natarajan, S., and Liu, J. W. S., Scheduling real-time, periodic job using imprecise results, *Proc. 8th Real-Time Systems Symp.*, San Francisco, CA, 1987.

[4] Shih, W.-K., Liu, J. W. S., Chung, J.-Y., and Gillies, D. W., Scheduling tasks with ready times and deadlines to minimize average error, *ACM Operating Syst. Rev.*, July 1989.

[5] Horn, W. A., Some simple scheduling algorithms, *Nav. Res. Logistics Q.*, 21, 177, 1974.

[6] Blazewicz, J., Scheduling preemptible tasks on parallel processors with information loss, *Tech. et Sci. Inf.*, 3, 415, 1984.

[7] Blazewicz, J. and Finke, G., Minimizing weighted execution time loss on identical and uniform processors, *Inf. Process. Lett.*, 24, 259, 1987.

[8] Leung, J. Y.-T., Minimizing total weighted error for imprecise computation tasks and related problems, in Leung, J. Y.-T., Ed., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, Boca Raton, FL, 2004, chap. 34.

[9] Potts, C. N. and Van Wassenhove, L. N., Single machine scheduling to minimize total late work, *Oper. Res.*, 40, 586, 1992.

[10] Potts, C. N. and Van Wassenhove, L. N., Approximation algorithms for scheduling a single machine to minimize total late work, *Oper. Res. Lett.*, 11, 261, 1992.

[11] Shih, W.-K., Liu, J. W. S., and Chung, J.-Y., Algorithms for scheduling imprecise computations with timing constraints, *SIAM J. Comput.*, 20, 537, 1991.

[12] Leung, J. Y.-T., Yu, V. K. M., and Wei, W.-D., Minimizing the weighted number of tardy task units, *Discrete Appl. Math.*, 51, 307, 1994.

[13] Ho, K. I.-J., Leung, J. Y.-T., and Wei, W.-D., Minimizing maximum weighted error for imprecise computation tasks, *J. Algorithms*, 16, 431, 1994.

[14] Ho, K. I.-J., Dual-criteria optimization problems for imprecise computation tasks, in Leung, J. Y.-T., Ed., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, Boca Raton, FL, 2004, chap. 35.

[15] Ho, K. I.-J. and Leung, J. Y.-T., A dual criteria preemptive scheduling problem for minimax error of imprecise computation tasks, *Int. J. Found. Comput. Sci.*, 15, 717, 2004.

[16] Lawler, E. L., A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs, *Ann. Oper. Res.*, 26, 125, 1990.

[17] Ho, K. I.-J., Leung, J. Y.-T., and Wei, W.-D., Scheduling imprecise computation tasks with 0/1-constraint, *Discrete Appl. Math.*, 78, 117–132, 1997.

[18] Gabow, H. N. and Tarjan, R. E., A linear-time algorithm for a special case of disjoint set union, *J. Comput. Syst. Sci.*, 30, 209, 1985.

45

Scheduling Malleable Tasks

Klaus Jansen
Kiel University

Hu Zhang
McMaster University

45.1 Introduction

Nowadays, parallel computing plays a key role in high-performance computing and it is gradually replacing traditional supercomputers. The computing units (processors) in parallel computing systems are linked by different types of networks [1]. The allocation of the computing resources to tasks is a crucial issue in these complex systems. Therefore there is a need to design efficient resource allocation algorithms. Unfortunately, many traditional scheduling algorithms are not able to fulfill this requirement, due to the large amount of data that needs to be transmitted through the network, as well as synchronization and other scheduling overhead required by the processors allotted to the same task. Thus, many parallel task (PT) models have been proposed to address this problem. One of these models is the *malleable task* (MT) model (sometimes also called *moldable task*) [2]. This is a promising model that has been used in real application [3]. In the MT model, the *processing time* of a task depends on the number of processors allotted to it. The communication overhead and synchronization is implicitly included in the processing time. The idea behind the MTs is to provide an alternative strategy to model the communication delays. An MT includes elementary operations (e.g., a numerical routine or a nested loop) with sufficient parallelism to be amenable for multiprocessor processing. Thus, a sequential task can be regarded as a special case of the MT model. The MT model has been used in realistic applications [3,4].

In general, there are three types of MT models based on the actual behavior of the PTs. In the first model (MT1), the processing time of an MT may be an arbitrary function of the number of processors allotted. In the second model (MT2), the processing time of an MT decreases as the number of processors allotted to it increases, while the *work function* (the product of the processing time and the number of processors) increases as the number of processors allotted increases. These assumptions are based on the well-known Brent's lemma [5], which states that the parallel execution of a task achieves some speedup if it is large enough, but does not lead to superlinear speedups. In the third model (MT3), the processing time has the same behavior as in MT2, while the *speedup function* (the reciprocal of the processing time function) in terms of the number of processors is concave. This is based on the real behavior of a class of massive parallel computers [6–8].

We shall study two different categories of MTs. In the first category, the MTs are *independent* MTs (called IMTs), that is, there is no dependence between the tasks. In the second category, there are *precedence constraints* between the MTs (called PCMTs). These dependencies are determined in advance from the

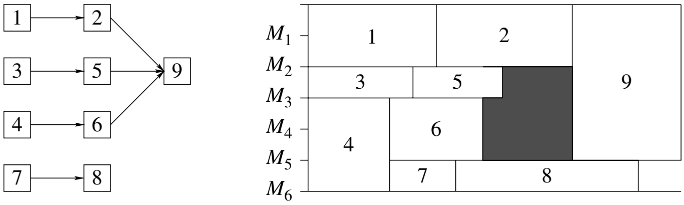


FIGURE 45.1 A precedence graph and a corresponding feasible schedule.

tasks' data flow. From the precedence constraints we construct a *precedence graph*, which is simply a directed acyclic graph. The vertices represent the set of MTs and the directed edges the set of precedence constraints among MTs (see Figure 45.1).

An instance of the MT scheduling problem consists of n independent or precedent-constrained MTs (MT1, MT2, or MT3) and m homogeneous (identical) processors. The problem is to find a feasible schedule with minimum *makespan* (or $C_{\max}$, which is the maximum completion time). In a feasible schedule, each task is executed simultaneously on all processors allotted to it until its completion time without interruption, and each processor executes at most one task at any time. For PCMTs, the precedence constraints must be satisfied, that is, a task cannot start until all its predecessors have been processed to completion.

Since classical PT scheduling problems are, in general, special cases of MT scheduling, some of their complexity results apply directly to MT scheduling. Thus, MT scheduling is $\mathcal{NP}$-hard [9]. IMT scheduling is known to be strongly $\mathcal{NP}$-hard even for five processors, but IMT scheduling for three processors is solvable in pseudopolynomial time [10]. The complexity of IMT scheduling on four processors is open. In contrast, PCMT scheduling is $\mathcal{NP}$-hard in the strong sense [10], and a lower bound for the approximation ratio is 4/3 [11]. Hence, there is interest in finding polynomial-time approximation algorithms for these problems.

Most existing approximation algorithms for MT scheduling are based on the two-phase approach initially proposed by Turek et al. [2]. In the first phase, an allotment problem is solved (approximately) such that each task is allotted a number of processors. Then in the second phase, the resulting non-MT scheduling is to be solved (approximately). It is clear that if we are able to approximate allotment within a ratio ρ and approximate the non-MT scheduling within a ratio μ, then we will have an r-approximation algorithm, where $r = \rho\mu$. On the basis of this strategy, the following two methodologies for IMT and PCMT scheduling have been used. The first one focuses on the non-MT scheduling phase (the second phase) and the other on the first phase (allotment problem). Let us discuss them further.

- Since we consider the makespan as the optimization criterion, the second phase for IMT scheduling is essentially the two-dimensional strip packing problem [12–15], which is to pack (without rotation) a set of rectangles with width at most 1 into a strip with width 1 and least possible height. In this approach, the allotment problem is usually formulated as a knapsack problem or one of its variants. Approximation algorithms in this category can be found in Refs. [16–21].
- For PCMT scheduling, the allotment problem is solved by dynamic programming for tree precedence constraints [22,23] or by approximation algorithms for the discrete time–cost trade-off problem [23–27]. The non-MT scheduling in the second phase is then solved by a variant of the list scheduling algorithm [28].

We shall review four approximation algorithms for MT scheduling in this chapter. In Section 45.2 we discuss a 3/2-approximation algorithm for IMT scheduling (MT2) [22], and then an AFPTAS for IMT scheduling (MT1) [21] in Section 45.3. A $(3+\sqrt{5})/2$-approximation algorithm for PCMT scheduling with tree precedence constraints (MT2) [23] is introduced in Section 45.4. Finally, we shall discuss in Section 45.5 a $(3 + \sqrt{5})$-approximation algorithm for PCMT scheduling with general precedence constraints (MT2) [23], and improved results for the MT2 and MT3 models [25–27].

45.1.1 Notations

An instance of the IMT and PCMT scheduling contains a set $\mathcal{T}$ of n tasks and a set of m identical processors. Each task J_j can be executed on any integer number $l \in \{1, \ldots, m\}$ of processors. It requires $p_j(l)$ units of time when being processed by l processors. In addition, a PCMT instance includes a directed precedence graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ represents PCMTs and $E \subseteq V \times V$ precedence constraints among PCMTs. If there is an edge $(i, j) \in E$, then task J_j cannot be processed before the completion of task J_i. Task J_i is called a *predecessor* of J_j, while J_j a *successor* of J_i. We denote by $\Gamma^-(j)$ and $\Gamma^+(j)$ the sets of the predecessors and successors of J_j, respectively. We also assume that $p_j(0) = \infty$, as no task J_j can be executed if there is no processor available. Furthermore, given a processing time $p_j(l)$, its *speedup* function is defined as $s_j(l) = p_j(1)/p_j(l)$, and its *work* function is $W_j(l) = lp_j(l)$. In addition, in some cases the work of a task J_j is also considered as a function of its processing time, denoted by $w_j(p_j(l)) = lp_j(l) = W_j(l)$.

In a schedule, each task J_j has two values associated with it: the starting time τ_j and the number of processors l_j allotted to task J_j. A task J_j is called *active* during the time interval from its starting time τ_j to its completion time $C_j = \tau_j + p_j(l_j)$. A schedule is *feasible* if at any time t, the number of active processors does not exceed the total number of processors, that is, $\sum_{j:t\in[\tau_j,C_j]} l_j \leq m$, and if there is no preemption or migration. For the PCMT case, a feasible schedule is further required to satisfy the precedence constraints $\tau_i + p_i(l_i) \leq \tau_j$ are fulfilled for all $i \in \Gamma^-(j)$.

We consider the following three models of MTs:

- MT1: There are no special conditions on the processing time or work.
- MT2: The following two constraints must be satisfied:
  - The processing time $p(l)$ of a MT J is nonincreasing in the number l of the processors allotted to it, that is, $p(l) \leq p(l')$, for $l \geq l'$
  - The work of a MT J is nondecreasing in the number l of the processors allotted to it, that is, $W(l) \leq W(l')$ for $l \leq l'$.
- MT3: The following two constraints must be satisfied:
  - The processing time $p(l)$ of a MT J is nonincreasing in the number l of the processors allotted to it, that is, $p(l) \leq p(l')$, for $l \geq l'$;
  - The speedup function of a MT J is concave in the number l of the processors allotted to it, that is, $p_j(1)/p_j(l) = s_j(l) \geq [(l - l'')s_j(l') - (l - l')s_j(l'')]/(l' - l'') = p_j(1)[(l - l'')/p_j(l') - (l - l')/p_j(l'')]/(l' - l'')$, for any $0 \leq l'' \leq l \leq l' \leq m$.

45.2 A 3/2-Approximation Algorithm for IMT Scheduling

The original strategy in Ref. [2] for IMT scheduling, in general, leads to an effective mechanism to design approximation algorithms. However, its power and its limitation are that the approximation ratios of this type of algorithms strongly depend on the approximation ratios for the strip packing algorithms. Thus, direct application of this approach can only lead to an absolute performance ratio of 2 based on Ref. [14]. Other approaches that avoid the strip packing bottleneck have been proposed, for example, the $(\sqrt{3} + \varepsilon)$-approximation algorithm for IMT scheduling (MT2) proposed in paper [18]. The approximation ratio was further improved to 3/2 [19]. This approximation algorithm for IMT scheduling (MT2) used the dual technique. An r-dual approximation algorithm for IMT scheduling (MT2) takes a real number d as input and

- either delivers a schedule of length at most rd;
- or gives a correct answer that there exists no schedule of length less than d.

Furthermore, a lower bound $\underline{C}_{\max}$ can be computed such that $\underline{C}_{\max} \leq C^*_{\max} \leq 2\underline{C}_{\max}$ [16]. Therefore, a binary search on this interval results is an $(r + \varepsilon)$-approximation solution provided there is an r-dual approximation algorithm for any $\varepsilon > 0$.

TABLE 45.1 The 3/2-Dual Approximation Algorithm for IMT Scheduling (MT2) in Ref. [19]

1. Remove the set $\mathcal{T}_S$ of small tasks whose processing times on one processor is at most $d/2$.
2. Find an allotment for the remaining tasks such that every task is alloted enough processors so that its processing time is at most d. Partition these remaining tasks into two sets $\mathcal{T}_1$ and $\mathcal{T}_2$, consisting of the tasks with processing times strictly greater than $d/2$ and at most $d/2$, respectively.
3. Apply basic transformations to generate a feasible two-level schedule (a two-level schedule for t is such that every task is either processed completely from time 0 to time t, or after time t). The original allotment for tasks may decrease by the transformations.
4. Schedule the tasks in $\mathcal{T}_S$.

The skeleton 3/2-dual approximation algorithm for IMT scheduling (MT2) is shown in Table 45.1. Later on we give implementation details for steps 2–4.

The following lemma appears in Ref. [19] and for brevity we state it here without a proof:

Lemma 45.1

If a schedule of length $3d/2$ exists for $\mathcal{T} \setminus \mathcal{T}_S$ with work $md - W_S$, then there exists an MT schedule of length at most $3d/2$, where W_S is the work of the set of tasks $\mathcal{T}_S$.

In steps 2 and 3 of the algorithm in Table 45.1, an allotment for tasks in $\mathcal{T} \setminus \mathcal{T}_S$ is constructed to satisfy the following constraints:

(C1) The total work of this allotment is at most $md - W_S$.
(C2) The set $\mathcal{T}_1$ of tasks with processing times strictly greater than $d/2$ use at most m processors. These tasks will be scheduled in level S_1 for time 0 to d.
(C3) The set $\mathcal{T}_2$ of tasks with processing times at most $d/2$ use at most m processors. These tasks will be scheduled in level S_2 from time d to $3d/2$.

Obviously, an allotment satisfying all the above three constraints for tasks in $\mathcal{T} \setminus \mathcal{T}_S$ leads to a two-level schedule of length at most $3/2d$. The set of tasks $\mathcal{T}_1$ is scheduled in level S_1 from time 0 to d, and the set of tasks $\mathcal{T}_2$ is to be scheduled in level S_2 from time d to $3d/2$. This schedule satisfies the conditions of Lemma 45.1. Therefore we know there is a 3/2-dual approximation solution to the IMT scheduling (MT2) problem. The remaining work is to find an allotment without violating constraints (C1)–(C3).

In the second step, an allotment for the tasks in $\mathcal{T} \setminus \mathcal{T}_S$ satisfying constraints (C1) and (C2) is computed by solving a knapsack problem. Denote by $l_j(x)$ the least number of processors needed to process task J_j so that it has a processing time at most x. Indeed, the knapsack problem to be solved is minimizing the total work of the tasks in $\mathcal{T} \setminus \mathcal{T}_S$ subject to the constraint that the tasks in $\mathcal{T}_1$ use a total number of processors at most m, that is,

$$\begin{aligned} \min_{\mathcal{T}_1 \subseteq \mathcal{T}} \quad & \sum_{J_j \in \mathcal{T}_1} W_j(l_j(d)) + \sum_{J_j \notin \mathcal{T}_1} W_j(l_j(d/2)) \\ \text{s.t.} \quad & \sum_{J_j \in \mathcal{T}_1} l_j(d) \leq m \end{aligned}$$

The knapsack problem is $\mathcal{NP}$-hard [9], but there allows a pseudopolynomial-time algorithm [29,30] by dynamic programming with running time $O(nC)$, where C is the capacity of the knapsack. In our case, the capacity C is m, so it can be solved in $O(mn)$ time. If solution to the above knapsack problem has total work greater than $md - W_S$, then there is no solution to the scheduling problem with makespan at most d, and the dual approximation algorithm gives a correct negative answer for value d. Otherwise, we show below that a feasible schedule can be constructed by a set of transformations. The above discussion can be summarized by the following lemma:

Lemma 45.2

If there exists a schedule of length at most d, then solving the knapsack formulation of the problem gives an allotment satisfying constraints (C1) and (C2).

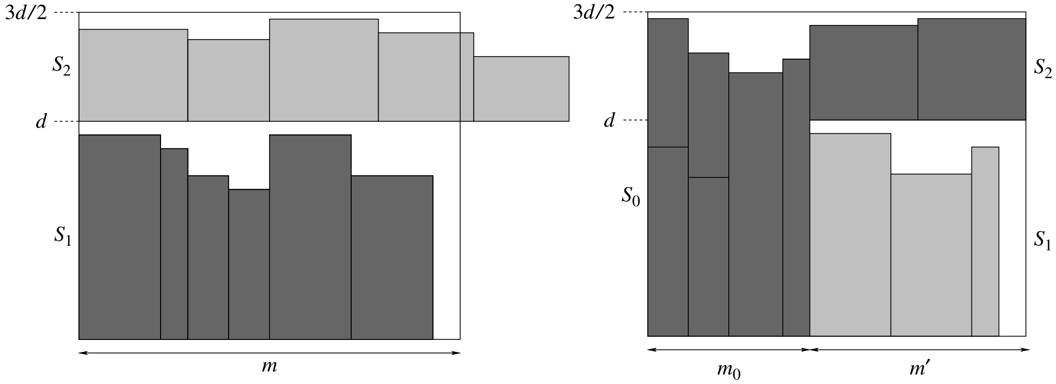


FIGURE 45.2 The two-level schedule from initial allotment to the final schedule by transformations.

From the allotment generated by the solution to the knapsack problem, we have a two-level schedule with tasks in $\mathcal{T}_1$ in level S_1 and the others $\mathcal{T}_2 = \mathcal{T} \setminus (\mathcal{T}_S \cup \mathcal{T}_1)$ in level S_2, provided that constraint (C3) is satisfied. But this condition might not hold (see the schedule on the left-hand side of Figure 45.2). Hence, we perform transformations to modify the shape of the two-level schedule to create a new area S_0 whose processors may be continuously busy in interval $[0, 3d/2]$. The transformations are as follows:

(T1) If a task J_j in S_1 has processing time at most $3d/4$ and is allotted to $l_j > 1$ processors, then allocate J_j to $l_j - 1$ processors and move it to S_0.

(T2) If J_j and J'_j in S_1 both have processing times at most $3d/4$ and both are allotted to one processor, then allocate J_j and J'_j to the same processor and move them to S_0. For the special case that J_j is the only remaining "sequential" task with processing time at most $3d/4$, it is put on top of a task in S_1 (if one exists) that has a processing time greater than $3d/4$, provided that the completion time of J_j does not exceed $3d/2$.

(T3) Denote by q the number of idle processors in S_1. If there exists a task J_j in S_2 such that its processing time on q processors is bounded by $3d/2$, then place J_j on l_j processors, where l_j is the smallest integer in $\{1, \ldots, m\}$ such that the processing time $p_j(l_j) \leq 3d/2$. Depending on the resulting processing time, J_j is either moved to S_0 or S_1.

The above transformations can be conducted in any order. It has been proven in Ref. [19] that the resulting schedule is feasible (see Figure 45.2) and the following lemma holds:

Lemma 45.3

An algorithm that performs the transformations (T1), (T2), and (T3) for a solution to the knapsack problem delivers a feasible schedule of length at most $3d/2$ *and total work* $md - W_S$.

Combining Lemmas 45.1 and 45.3, together with the binary search strategy, we have a $(3/2 + \varepsilon)$-approximation algorithm for IMT scheduling (MT2) [19]. This result is summarized in the following theorem:

Theorem 45.1

The algorithm described above is a $(3/2 + \varepsilon)$*-approximation algorithm for IMT scheduling (MT2).*

45.3 An AFPTAS for IMT Scheduling

Given an allotment of IMTs, non-MT scheduling is in fact the strip packing problem. On the basis of this idea, a 2-approximation algorithm is given in Refs.[16,17] using the 2-approximation algorithm for strip packing given in Ref. [14]. Since there is an AFPTAS for strip packing [15], a natural question is whether there exists an AFPTAS for IMT scheduling. In this section we discuss the AFPTAS given in Ref. [21] for IMT scheduling (MT1) with processing times at most 1.

The main ideas of the AFPTAS are as follows: First an approximate preemptive schedule with migration for IMT scheduling (MT1) is constructed via linear programming. Then the preemptive schedule is converted into a nonpreemptive schedule by computing a unique allotment for almost all MTs using a new rounding technique. Let us now discuss this in detail.

In the preemptive model, each task can be interrupted at any time before its completion at no cost and can be resumed later on. In addition, migration is allowed for this preemptive schedule, which means that a task may be assigned to different processor sets during its different execution phases [10,31,32]. The preemptive scheduling problem can be formulated as the following linear program [33]:

$$\begin{aligned} \min \quad & \sum_{f\in F} x_f \\ \text{s.t.} \quad & \sum_{l} \frac{1}{p_j(l)} \sum_{f\in F:|f^{-1}(j)|=l} x_f \geq 1, \quad j = 1, \ldots, n \\ & x_f \geq 0, \quad \forall f \in F \end{aligned} \tag{45.1}$$

The set F denotes the set of all configurations. The variable x_f indicates the length of configuration f in the schedule, and $|f^{-1}(j)|$ is the number of processors allotted to task J_j in configuration f. Clearly, the optimal objective function value of Eq. (45.1), that is, the length of the optimal preemptive schedule with migration, is a lower bound of the length of the optimal nonpreemptive IMT schedule.

The linear program (45.1) can be solved approximately by performing binary search on the optimum value and testing at each step the feasibility of a system of (in)equalities for a given $g \in [d_{\max}, nd_{\max}]$, where $d_{\max} = \max_{j=\{1,\ldots,n\}} d_j$ and $d_j = \min_l p_j(l)$. Notice that the length of an optimal preemptive schedule is at least $d_{\max}$ and at most $nd_{\max}$. Now the system of inequalities is

$$\sum_{l} \frac{1}{p_j(l)} \sum_{f\in F:|f^{-1}(j)|=l} x_f \geq 1, \quad j = 1, \ldots, n, \quad (x_f)_{f\in F} \in B$$

where

$$B = \left\{ (x_f)_{f\in F} | \sum_{f\in F} x_f = g, \quad x_f \geq 0, \quad f \in F \right\}$$

This test can be performed approximately by computing an approximate solution to the following problem:

$$\lambda^* = \max \left\{ \lambda | \sum_{l} \frac{1}{p_j(l)} \sum_{f\in F:|f^{-1}(j)|=l} x_f \geq \lambda, \quad j = 1, \ldots, n, \quad (x_f)_{f\in F} \in B \right\} \tag{45.2}$$

which can be viewed as a fractional covering problem with convex set B and n covering constraints. Thus, the $(1-\varepsilon)$-approximation algorithm for fractional covering problems in Ref. [34] can be used. It requires a block solver that for any price vector $y \in \mathbb{R}_+^n$ computes an approximate solution of

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} \sum_{l} \frac{y_j}{p_j(l)} \cdot x_{jl} \\ \text{s.t.} \quad & \sum_{j=1}^{n} \sum_{l} l \cdot x_{jl} \leq m \\ & \sum_{l} x_{jl} \leq 1, \quad j = 1, \ldots, n \\ & x_{jl} \in \{0, 1\}, \quad l = 1, \ldots, m, \quad j = 1, \ldots, n \end{aligned} \tag{45.3}$$

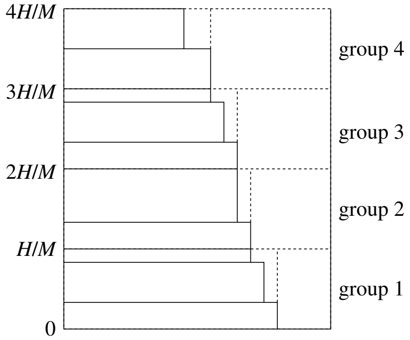


FIGURE 45.3 The stack packing for rounding.

This is indeed the multiple-choice knapsack problem (a generalized knapsack problem with different choices for tasks). Lawler [35] presented a $(1 - \varepsilon)$-approximation algorithm for this problem. Hence, we have developed an FPTAS for the preemptive IMT scheduling (MT1) with migrations. In the second phase of the algorithm, the resulting preemptive schedule is converted into a nonpreemptive schedule.

The conversion phase consists of two steps. First, a unique processor allotment for almost all tasks is generated. Subsequently, a new rounding technique is applied and a constant number of tasks with nonunique processor numbers are removed. Then an instance of strip packing with one rectangle for any remaining tasks is defined, and its approximate solution leads to an approximate solution to IMT scheduling (MT1).

Let $p_{\max} = \max_j \max_l p_j(l)$. For each task $J_j \in \mathcal{T}$ and each processor number l, the fraction is defined as $x_{j,l} = \sum_{f \in F:|f^{-1}(j)|=l} x_f / p_j(l) \geq 0$. Without loss of generality, we assume that $\sum_l x_{j,l} = 1$. Then an instance of strip packing is constructed, with rectangles of height $x_{j,l} p_j(l)$ and width l, for $x_{j,l} > 0$. Denote by L_w the set of rectangles with width $l > \varepsilon' m$ and L_n the remaining rectangles, where ε' is defined later. Similar to the AFPTAS for strip packing in Ref. [15], for the wide rectangles in L_w, an instance of stack packing is constructed such that the total height of the wide rectangles $H = H(L_w)$ is the height of the stack, and the stack is divided into M groups, where M is defined later (see Figure 45.3). Let $W_{j,l}$ be the set of widths in group i corresponding to task J_j. The total area of the narrow tasks is $A_n = \sum_{j=1}^{n} \sum_{l \leq \varepsilon' m} x_{j,l} p_j(l) l$, and they are placed in group 0. For each group $i \in \{0, \ldots, M\}$ and task J_j, let $z_{j,i} = \sum_{w:w \in W_{j,i}} y_{j,i}(w)$ be the fraction of task J_j executed in group i. Denote by a_i and by b_i the smallest and the largest widths of group i, respectively. The following steps are performed to obtain unique processor numbers for almost all the tasks:

(1) For each group i and task J_j with at least two widths in group i, compute the smallest processing time $p_j(l)$ among all processor numbers $l \in [a_i, b_i]$. Let $l_{j,i}$ be such a processor number. Now place the rectangles corresponding to task J_j in group i by $(z_{j,i} p_j(l_{j,i}), l_{j,i})$.

(2) For each task J_j with at least two widths in group 0, compute the smallest area $p_j(l)l$ among all small processor numbers $l \leq \varepsilon' m$. Let $l_{j,0}$ be such a processor number. Then replace all rectangles corresponding to task J_j in group 0 by $(z_{j,0} p_j(l_{j,0}), l_{j,0})$.

(3) Round all tasks over the groups using a general assignment problem:

$$\begin{aligned}
&\sum_{j=1}^{n} z_{j,0} p_j(l_{j,0}) l_{j,0} \leq A_n \\
&\sum_{j=1}^{n} z_{j,i} p_j(l_{j,i}) \leq H/M, \quad i = 1, \ldots, M \\
&\sum_{i=0}^{M} z_{j,i} = 1, \qquad j = 1, \ldots, n \\
&z_{j,i} \geq 0, \qquad j = 1, \ldots, n, \quad i = 1, \ldots, M
\end{aligned} \tag{45.4}$$

TABLE 45.2 The Algorithm to Convert a Preemptive Schedule into a Nonpreemptive Schedule in Ref. [21]

(0) Set $\delta \leq \min\{1, \varepsilon/8\}$, $\varepsilon' = \delta/(\delta+2)$, $M = 1/\varepsilon'^2$;
(1) compute the values $x_{j,l} = \sum_{f \in F:|f^{-1}(j)|=l} x_f / p_j(l) \in [0, 1]$ for each task $T_j \in T$ and l;
(2) construct an instance of strip packing with rectangles $(x_{j,l}\, p_j(l), l)$ for $x_{j,l} > 0$, and let $L_w = \{(x_{j,l}\, p_j(l), l) | l > \varepsilon' m\}$ and $L_n = \{(x_{j,l}\, p_j(l), l) | l \leq \varepsilon' m\}$.
(3) apply the rounding technique to obtain a strip packing with instance $L'_w = \{(p_j(l_{j,i}), l_{j,i}) | z_{j,i} = 1, i > 0\}$ and $L'_n = \{(p_j(l_{j,0}), l_{j,0}) | z_{j,0} = 1\}$ and set $F = \{T_j | z_{j,i} \in (0,1)$ for at least one $i \in \{0, \ldots, M\}\}$;
(4) construct instance $\sup(L'_w)$ (via strip packing in Ref. [15]) with a constant number M of distinct widths;
(5) solve fractional strip packing for $\sup(L'_w)$ approximately with ratio $(1+\delta)$ by the algorithm in Ref. [34] and round the solution to obtain only M nonzero variables x_j;
(6) place the wide rectangles of L'_w into the space generated by the nonzero variables x_j;
(7) insert the narrow rectangles of L'_n using modified next fit decreasing height [13,15];
(8) schedule the tasks in F at the end of the schedule.

This formulation is closely related to the one for scheduling (independent) tasks on unrelated machines [36]. Thus, one can now round the variables $z_{j,i}$ such that there are at most M fractional variables [36]. For tasks with integer variables, unique processor numbers are allotted. The remaining tasks are executed at the end of the schedule with processing time at most M. We now obtain a rectangle packing instance with a set $L'_w = \{(p_j(l_{j,i}), l_{j,i}) | z_{j,i} = 1,\ i > 0\}$ of wide rectangles and a set $L'_n = \{(p_j(l_{j,0}), l_{j,0}) | z_{j,0} = 1\}$ of narrow rectangles.

For the next step we use an instance sup(L) for a given set L of rectangles [15]. First construct a stack packing for L with M groups. Then rounding each rectangle in group i up to b_i (up to m for the first group) generates sup(L). We say that $L \leq L'$ if the stack associated to L viewed as a region in the plane is contained in the stack associated to L'. Our algorithm computes $\sup(L'_w)$ (by producing a new stack packing). In this way an instance with at most M distinct widths is generated. We note that the height H' of the new stack packing is bounded by H.

The algorithm to convert the preemptive schedule into nonpreemptive schedule is shown in Table 45.2. Hence, for any $\varepsilon > 0$, the AFPTAS for IMT scheduling (MT1) works as follows: First we set $\delta = \min\{1, \varepsilon/8\}$; afterwards we compute a $(1+\delta)$-approximate preemptive schedule. Then we convert the preemptive schedule into a nonpreemptive schedule.

Theorem 45.2

If the maximum processing time is at most 1, *then there exists an AFPTAS for IMT scheduling (MT1)* .

To improve the running time [21], a preprocessing step was added together with a speedup technique for the binary search and the solution to the linear programming problem (see Ref. [21] for details). A simplified AFPTAS for IMT scheduling (MT2) is also presented in Ref. [21].

45.4 An Approximation Algorithm for PCMT Scheduling with Tree Precedence

We study a special case of PCMT scheduling, where the precedence graph is a tree, a series parallel graph [37] or a bounded width graph. In Ref. [22] the schedule delivered by their algorithm is carefully analyzed. The bound of the increase of the schedule length in the second phase (non-MT schedule) is estimated and their algorithm is shown a 4-approximation algorithm for PCMT scheduling with tree precedence. The performance ratio is further improved to $(3+\sqrt{5})/2 \approx 2.61803$ in Ref. [23].

As in Ref. [21], the approximation algorithm in Ref. [23] for PCMT scheduling with tree precedence has two phases. In the first phase, the allotment problem is solved such that each task is allotted a certain number of processors. In the second phase, for the given allotment (may be slight different from the solution to the allotment problem), a non-MT schedule is obtained by classical scheduling algorithms.

TABLE 45.3 Algorithm **LIST**

```
LIST ($J$, $m$, $\alpha'$, $\mu$)
• generate new allotment $\alpha$: $l_j = \min\{l'_j, \mu\}$ for $j \in \{1, \ldots, n\}$;
• SCHEDULED $= \emptyset$;
• if SCHEDULED $\neq \mathcal{T}$ then
    ◦ READY $= \{J_j | \Gamma^-(j) \subseteq\}$ SCHEDULED;
    ◦ compute the earliest possible starting time under $\alpha$ for all tasks in READY;
    ◦ schedule the task $J_j \in$ READY with the smallest earliest starting time;
    ◦ SCHEDULED $=$ SCHEDULED $\cup \{J_j\}$;
• end
```

The allotment problem in this case is to find for each task a number of processors, to minimize the following cost:

$$\text{cost} = \max\{L,\ W/m\} \tag{45.5}$$

where L is the length of a *critical path* (the longest path in the precedence graph under the allotment), and W the sum of the works of all tasks.

We first study the second phase, that is, given an allotment, to schedule PCMTs according to their precedence. In fact in Refs. [23,26,27], a variant of the list scheduling algorithm [28] is performed (Table 45.3). Denote by l'_j the number of processors allotted to a task J_j in the allotment α' generated in the first phase (i.e., a solution to the allotment problem). The algorithm **LIST** constructs a new allotment α with a parameter $\mu \in \{1, \ldots, \lfloor (m+1)/2 \rfloor\}$ defined later, and then schedules all tasks in a greedy way.

Denote by α^* the optimal (fractional) solution to the allotment problem, and by L^* and W^* its critical path length and total work, respectively. In addition, let L and W denote the critical path length and total work of the final schedule corresponding to allotment α.

The following bound holds by using Eq. (45.5) for α^*:

$$\max\{L^*,\ W^*/m\} \leq C^*_{\max} \tag{45.6}$$

where $C^*_{\max}$ is the length of α^*. Clearly, it is also a lower bound of an optimal PCMT schedule. Suppose that we have already a λ-approximate solution of the allotment problem, that is,

$$\max\{L',\ W'/m\} \leq \lambda \max\{L^*,\ W^*/m\} \tag{45.7}$$

Our goal is to find the relation between the makespan of the final schedule and that of the allotment in the first phase. Since in the second phase, each task J_j is allotted $l_j \leq l'_j$ processors, according to the property of MT2, the work does not increase. Therefore

$$W \leq W' \leq m\lambda C^*_{\max} \tag{45.8}$$

In the final schedule, the time interval $[0, C_{\max}]$ consists of three types of time slots. In the first type of time slots, at most $\mu - 1$ processors are busy. In the second type of time slots, at least μ and at most $m - \mu$ processors are busy. In the third type at least $m - \mu + 1$ processors are busy. Denote by T_1, T_2, and T_3 the sets of the three types of time slots, and by $|T_i|$ the overall lengths for $i \in \{1, 2, 3\}$. In the case that $\mu = (m+1)/2$ and m odd, $T_2 = \emptyset$. In other cases all three types of time slots may exist. Clearly,

$$C_{\max} = |T_1| + |T_2| + |T_3| \tag{45.9}$$

Since during the first (respectively, the second and the third) type of time slots, at least one of the (respectively, μ and $m - \mu + 1$) processors is busy, we have

$$W \geq |T_1| + \mu|T_2| + (m - \mu + 1)|T_3| \tag{45.10}$$

Lemma 45.4

$|T_1| + \mu|T_2|/m \leq L'$.

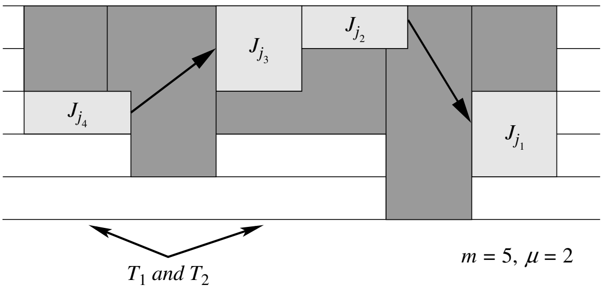


FIGURE 45.4 An example of the "heavy" path.

Proof
We construct a "heavy" directed path $\mathcal{P}$ in the final schedule. The last task in the path $\mathcal{P}$ is any multiprocessor task J_{j_1} that completes at time $C_{\max}$ (the makespan of the final schedule). After we have defined the last $i \geq 1$ tasks $J_{j_i} \to J_{j_{i-1}} \to \cdots \to J_{j_2} \to J_{j_1}$ on the path $\mathcal{P}$, we can determine the next task $J_{j_{i+1}}$ as follows: Consider the latest time slot t in $T_1 \cup T_2$ that is before the starting time of task J_{j_i} in the final schedule. Let V' be the set of task J_{j_i} and its predecessor tasks that start after time t in the schedule. Since during time slot t at most $m - \mu$ processors are busy, and since at most μ processors are allotted to any task in V', none of the tasks in V' can be ready for execution during the time slot t. Therefore for every task in V' some predecessor is being executed during the time slot t. Then we select any predecessor of task J_{j_i} that is running during slot t as the next task $J_{j_{i+1}}$ on the path $\mathcal{P}$. This search procedure stops when $\mathcal{P}$ contains a task that starts before any time slot in $T_1 \cup T_2$. An example of the "heavy" path is illustrated in Figure 45.4. Now we examine the stretch of processing time for all jobs in $\mathcal{P}$ in the rounding procedure of the first phase and in the new allotment α of the second phase.

Consider a task J_j in the resulting path $\mathcal{P}$. If in the final schedule $l_j < \mu$, then in the first phase $l'_j = l_j < \mu$. If $l_j = \mu$, then $\mu \leq l'_j \leq m$. Since $l_j p_j(l_j) \leq l'_j p_j(l'_j)$ by the property of MT2, $p_j(l'_j)/p_j(l_j) \geq \mu/l'_j \geq \mu/m$. With the construction of the directed path $\mathcal{P}$, it covers all time slots in $T_1 \cup T_2$ in the final schedule. In addition, denote by $L'(\mathcal{P})$ the length of path $\mathcal{P}$ in allotment α'. The tasks during time slots in T_1 contribute a total length of at least $|T_1|$ to $L'(\mathcal{P})$, and tasks in T_2 contribute at least $\mu|T_2|/m$ to $L'(\mathcal{P})$. Because $L'(\mathcal{P}) \leq L'$, the lemma follows. □

Combining the above bounds, the following theorem holds:

Theorem 45.3

If there exists a λ-approximation algorithm for the allotment problem, then there exists a λr-approximation algorithm for PCMT scheduling (MT1), where $r = \min_{1\leq\mu\leq\lfloor(m+1)/2\rfloor} \max\{m/\mu, (2m-\mu)/(m-\mu+1)\}$.

Proof
Multiplying Eq. (45.9) by $m - \mu + 1$ and subtracting Eq. (45.10) from it yields

$$(m - \mu + 1)C_{\max} \leq W + (m - \mu)|T_1| + (m - 2\mu + 1)|T_2| \tag{45.11}$$

We consider two cases. In the first case, $m/\mu \leq (2m-\mu)/(m-\mu+1)$. Thus, $r = (2m-\mu)/(m-\mu+1)$ and $(m-2\mu+1) \leq \mu(m-\mu)/m$. Substituting this into Eq. (45.11), using Lemma 45.4, and Eq. (45.6)–(45.8) we have $(m - \mu + 1)C_{\max} \leq (2m - \mu)\lambda C^*_{\max}$, and the resulting schedule is indeed a λr-approximate solution. The analysis of the second case is analogous. □

Furthermore, it can be proven that for all $m \geq 2$, μ equals either the integer above or the integer below $(3m - \sqrt{5m^2 - 4m})/2$, and $r < (3 + \sqrt{5})/2 \approx 2.61803$. As m tends to infinity, μ tends to $(3 - \sqrt{5})m/2 \approx 0.38196m$, and r tends to $(3 + \sqrt{5})/2$.

We have now established the relation between the approximation ratio for the allotment problem and PCMT scheduling (MT2). When the precedence graph is a tree, a series–parallel graph, or a bounded width graph, the allotment problem can be solved approximately with ratio $1 + \varepsilon$ (for any $\varepsilon > 0$) by dynamic programming [22,23]. In this way, we have already obtained a 2.61803-approximation algorithm for PCMT with tree precedence (MT2).

45.5 Approximation Algorithms for PCMT Scheduling

In the case of arbitrary precedence graphs, the allotment problem is related to the *discrete time–cost trade-off* problem [24,38], which is $\mathcal{NP}$-complete [39]. In fact, the approximation algorithm in Ref. [24] for the discrete time–cost trade-off problem is employed to solve the allotment problem in Ref. [23] for developing a 5.23606-approximation algorithm for PCMT scheduling (MT2).

An instance of the discrete time–cost trade-off problem is a *project* given by a finite set J of *activities* with a *partial order* $(J, \prec)$ on the set of activities. All activities have to be executed in accordance with the precedence constraints given by the partial order. Each activity $J_j \in J$ has a set of feasible *durations* $\{d_{j_1}, \ldots, d_{j_{k(j)}}\}$ sorted in a nondecreasing order, and has a nonincreasing nonnegative *cost* function $c_j : \mathbb{R}_+ \to \mathbb{R}_+ \cup \{\infty\}$, where $c_j(x_j)$ is the amount paid to run J_j with duration x_j.

Skutella [24] presented approximation algorithms for the above problems, in particular, an algorithm for the bicriteria problem such that $c(x) < B/(1-\rho)$ and $t(x) \leq L/\rho$ for a fixed $\rho \in (0, 1)$. The budget problem can be formulated as the following integer linear program:

$$\begin{array}{lll}
\min & L & \\
\text{s.t.} & 0 \leq C_j \leq L, & \text{for all } j \\
& C_i + x_j \leq C_j, & \text{for all } i \in \Gamma^-(j) \text{ and all } j \\
& c(x) = \displaystyle\sum_{j=1}^{n} c_j(x_j) \leq B & \\
& x_j \in \{d_{j_1}, \ldots, d_{j_{k(j)}}\}, & \text{for all } j
\end{array} \tag{45.12}$$

Here C_j is the completion time of activity J_j, $c_j(x_j)$ the cost of the duration x_j, and d_{j_p} the pth feasible duration of activity J_j. The first set of constraints shows that completion times of any tasks are bounded by the project length. The second set is related to the precedence constraints. In addition, the third set of constraint means that the total cost should be bounded by the budget B.

To solve it, first a "reduced" cost function $\hat{c}$ is set such that for any duration d_{j_l}, $\hat{c}_j(d_{j_l}) = c_j(d_{j_l}) - c_j(d_{j_{k(j)}})$. Since $d_{j_{k(j)}}$ is the maximum duration for activity J_j, $c_j(d_{j_{k(j)}})$ is the minimum over all durations for activity J_j and the "reduced" cost $\hat{c}$ is also positive. The amount of $P = \sum_{j=1}^{n} c_j(d_{j_{k(j)}})$ is called the "fixed" cost. In the second step, the "reduced" instance is transformed into a two-duration instance such that each given "virtual" activity has only at most two feasible durations. For any activity J_j, the first "virtual" activity J_{j_1} has only two fixed feasible durations $s_j(1) = t_j(1) = d_{j_1}$, and the corresponding "virtual" costs are $\bar{c}_j(s_j(1)) = \bar{c}_j(t_j(1)) = 0$. Then for each $1 < i \leq k(j)$, activity J_{j_i} has a duration $x_{j_i} \in \{s_j(i) = 0, t_j(i) = d_{j_i}\}$, and the corresponding "virtual" costs are $\bar{c}_{j_i}(s_j(i)) = \hat{c}_j(d_{j_{i-1}}) - \hat{c}_j(d_{j_i})$ and $\bar{c}_{j_i}(t_j(i)) = 0$. Thus the activity J_j can be modeled as $k(j)$ parallel "virtual" activities, and there is a canonical mapping of feasible durations x_j for activity J_j to tuples of feasible durations $x_{j_1}, \ldots, x_{j_{k(j)}}$ for "virtual" activities $J_{j_1}, \ldots, J_{j_{k(j)}}$ such that the duration of the activity J_j is the maximum over all durations of the corresponding "virtual" activities and the cost of J_j is the sum of the costs of the "virtual" activities, that is, $x_j = \max\{x_{j_1}, \ldots, x_{j_{k(j)}}\}$ and $\hat{c}_j(x_j) = \sum_{i=1}^{k(j)} \bar{c}_{j_i}(x_{j_i})$. Moreover, this mapping is bijective if we restrict ourselves without loss of generality to tuples of durations $x_{j_1}, \ldots, x_{j_{k(j)}}$ satisfying $x_{j_i} = t_j(i)$ if $t_j(i) \leq \max\{x_{j_1}, \ldots, x_{j_{k(j)}}\}$. In this way we have obtained a two-duration instance such that each activity has at most two feasible durations, and the solution of this instance can be transformed into a solution of the "reduced" instance.

Finally, we consider the linear relaxation of the two-duration instance, where for each "virtual"activity J_{j_i}, the "virtual" cost function is linear and nonincreasing within the interval $[s_j(i), t_j(i)]$ as follows:

$$\bar{c}_{j_i}(y) = \begin{cases} \infty, & \text{if } y < s_j(i) \\ \dfrac{t_j(i) - y}{t_j(i) - s_j(i)} \bar{c}_{j_i}(s_j(i)) + \dfrac{y - s_j(i)}{t_j(i) - s_j(i)} \bar{c}_{j_i}(t_j(i)), & \text{if } s_j(i) \leq y \leq t_j(i) \\ \bar{c}_{j_i}(t_j(i)) = 0, & \text{if } y \geq t_j(i) \end{cases} \tag{45.13}$$

Therefore we are able to find a fractional solution of the linear relaxation of the two-duration instance. To obtain a feasible (integer) solution of Eq. (45.12), we need to take some rounding technique. For a given $\rho \in (0, 1)$, if for a "virtual" activity J_{j_i} the fractional solution $x_{j_i} \in [0, \rho d_{jk(j)})$, we round it to $\bar{x}_{j_i} = 0$. Otherwise if $x_{j_i} \in [\rho d_{jk(j)}, d_{jk(j)}]$ we round it to $\bar{x}_{j_i} = d_{jk(j)}$, where $\bar{x}$ is the rounded solution. In this way, Skutella [24] showed that $\hat{c}(x) \leq (B - P)/(1 - \rho)$ and $C_{\max} \leq L/\rho$, where $B - P$ is the optimal cost for the linear relaxation of Eq. (45.12) with a deadline L.

In Ref. [23], the above algorithm is applied together with binary search to solve the allotment problem approximately. The rounding parameter $\rho = 1/2$ is set to obtain a 2-approximation algorithm for the allotment problem. According to Theorem 45.3, the following theorem holds:

Theorem 45.4

There exists a 5.23606*-approximation algorithm for PCMT scheduling (MT2).*

45.5.1 Improved Approximation Algorithm for PCMT Scheduling

Jansen and Zhang [27] observed the following two facts: First, solving the discrete time–cost trade-off problem for the two optimization criteria with the same ratio in the first phase in Ref. [23] does not necessarily lead to the best possible ratio for the whole algorithm. Second, the "fixed" cost in solving the discrete time–cost trade-off problem does not change during the rounding procedure. Thus, they further improved the approximation ratio for PCMT scheduling (MT2) to 4.730598.

For any MT J_j of MT2, we denote by $w(\cdot)$ the work function in processing time, that is, $w_j(p_j(l)) = W_j(l)$, and by x_j the fractional duration of the allotment problem (or the processing time). The new linear relaxation of the allotment problem is

$$\begin{array}{lll} \min & C = \max\{L, W/m\} & \\ \text{s.t.} & 0 \leq C_j \leq L, & \text{for all } j \\ & C_j + x_k \leq C_k, & \text{for all } j \text{ and } k \in \Gamma^+(j) \\ & x_j \leq p_j(1), & \text{for all } j \\ & x_{j_i} \leq x_j, & \text{for all } j \text{ and } i = 1, \ldots, m \\ & 0 \leq x_{j_i} \leq p_j(i), & \text{for all } j \text{ and } i = 2, \ldots, m \\ & x_{j_1} = p_j(m), & \text{for all } j \\ & \hat{w}_j(x_j) = \displaystyle\sum_{i=1}^{m} \bar{w}_{j_i}(x_{j_i}), & \text{for all } j \\ & P = \displaystyle\sum_{j=1}^{n} p_j(1) & \\ & \displaystyle\sum_{j=1}^{n} \hat{w}_j(x_j) + P \leq W & \end{array} \tag{45.14}$$

where the "virtual" work function $\bar{w}_j(x_{j_m}) = 0$, and for all j and $i = 1, \ldots, m - 1$:

$$\bar{w}_{j_i}(x_{j_i}) = [W_j(i+1) - W_j(i)](p_j(i) - x_{j_i})/p_j(i) \tag{45.15}$$

The rounding technique in Ref. [24] is applied here, and the following inequality holds:

$$L' \leq L^*/\rho \quad \text{and} \quad (W' - P) \leq (W^* - P)/(1 - \rho) \tag{45.16}$$

Furthermore, the bounds (45.6), (45.8), and (45.9) still hold. We can prove the following bound similar to Eq.(45.4):

Lemma 45.5

$\rho|T_1| + \min\{\mu/m, \rho\}|T_2| \leq C^*_{\max}$.

Denote by $x_i = |T_i|/C^*_{\max}$ the normalized lengths of the ith type of time slots, it can be shown that the approximation ratio is the objective value of the following min–max nonlinear program:

$$\begin{array}{rl} \min_{\mu,\rho} \max_{x_1,x_2} & \dfrac{x_1(m-\mu)(1-\rho) + x_2(1-\rho)(m-2\mu+1) + m}{(m-\mu)(1-\rho)+1} \\ \text{s.t.} & \rho x_1 + \min\{\rho, \mu/m\}x_2 \leq 1 \\ & x_1 + x_2(\mu(1-\rho)+\rho) \leq m \\ & x_1, x_2 \geq 0 \\ & \rho \in (0,1) \\ & \mu \in \{1, \ldots, \lfloor (m+1)/2 \rfloor\} \end{array} \tag{45.17}$$

A complicated analysis is needed to solve Eq. (45.17). It can be shown that to obtain the optimal value, one need to find analytic roots of a polynomial of degree 6, where the coefficients are polynomials of m. Unfortunately this is impossible in general. Thus in Ref. [27] the parameters are set as $\rho = 0.43$ and $\mu = (93m - \sqrt{4349m^2 - 4300m})/100$. Hence they showed that

Theorem 45.5

There exists an algorithm for PCMT scheduling (MT2) with an approximation ratio

$$r \leq \frac{100}{43} + \frac{100}{43}\frac{(43m-100)(57\sqrt{4349m^2-4300m} - 399m - 4300)}{139707m^2 - 174021m - 184900} \leq \frac{100}{43} + \frac{100(\sqrt{4349}-7)}{2451}$$
$$\approx 4.730598.$$

It is also proved that when m tends to infinity, the optimal asymptotic choices are $\rho = 0.430991$ and $\mu \to 0.270875m$. In this case the ratio $r \to 4.730577$.

45.5.2 Approximation Algorithm for PCMT Scheduling (MT3)

Jansen and Zhang [26] also studied the scheduling problem for model MT3, which is a generalization of the continuous model by Refs. [6–8]. The ideas are similar to those for MT2 in Ref. [27].

First, one can show that MT3 is a special case of MT2 where the work function is convex in the processing time. Then a piecewise linear program with convex constraints is developed for the relaxed allotment problem, which is polynomial-time solvable. Finally, the variant of the list scheduling algorithm is applied to obtain a feasible schedule.

The relaxed allotment problem can be formulated as

$$\begin{array}{rll} \min & C = \max\{L, W/m\} & \\ \text{s.t.} & C_i + x_j \leq C_j, & \text{for all } j \text{ and } i \in \Gamma^-(j) \\ & 0 \leq C_j \leq L, & \text{for all } j \\ & W/m = \displaystyle\sum_{j=1}^{n} w_j(x_j)/m & \\ & x_j \in [p_j(m), p_j(1)], & \text{for all } j \end{array} \tag{45.18}$$

where the continuous work function is defined as follows: If $x = p_j(l)$ for $l = 1, \ldots, m$, then $w_j(x_j) = w_j(p_j(l))$. If $x \in (p_j(l+1), p_j(l))$ for $l = 1, \ldots, m-1$, then

$$w_j(x_j) = \frac{w_j(p_j(l+1)) - w_j(p_j(l))}{p_j(l+1) - p_j(l)} x_j + \frac{w_j(p_j(l))p_j(l+1) - w_j(p_j(l+1))p_j(l)}{p_j(l+1) - p_j(l)} \tag{45.19}$$

In the interval $[p_j(l+1), p_j(l)]$, we define a critical point l_c such that $l_c = l + 1 - \rho$ for the rounding parameter $\rho \in [0, 1]$. The processing time $p_j(l_c)$ is given by $p_j(l_c) = p_j(l+1-\rho) = \rho p_j(l) + (1-\rho)p_j(l+1)$, and its work is $w_j(p_j(l_c)) = (1-\rho)w_j(p_j(l+1)) + \rho w_j(p_j(l)) = (1-\rho)(l+1)p_j(l+1) + \rho l p_j(l)$. We apply the following rounding technique for the fractional solution to Eq. (45.18): If $x_j^* \geq p_j(l_c)$ it is rounded up to $p_j(l)$, and otherwise rounded down to $p_j(l+1)$.

We have the following new bounds:

Lemma 45.6

For any job J_j, in the allotment α' its processing time $p_j(l'_j) \leq 2x_j^*/(1+\rho)$, and its work $w_j(p_j(l'_j)) = l'_j p_j(l'_j) \leq 2l_j^* x_j^*/(2-\rho) = 2w_j(x_j^*)/(2-\rho)$, where x_j^* is the optimal solution to Eq. (45.18).

Lemma 45.7

$(1+\rho)|T_1|/2 + \min\{\mu/m, (1+\rho)/2\}|T_2| \leq C^*_{\max}$.

It is shown that the approximation ratio is the objective value of the following min–max nonlinear program [26]:

$$\begin{aligned} \min_{\mu,\rho} \max_{x_1,x_2} \quad & \frac{2m/(2-\rho) + (m-\mu)x_1 + (m-2\mu+1)x_2}{m-\mu+1} \\ \text{s.t.} \quad & (1+\rho)x_1/2 + \min\{\mu/m, (1+\rho)/2\}x_2 \leq 1 \\ & x_1, x_2 \geq 0 \\ & \rho \in [0, 1] \\ & \mu \in \{1, \ldots, \lfloor (m+1)/2 \rfloor\} \end{aligned} \tag{45.20}$$

Again, the optimal solution cannot be obtained analytically in general. Hence the parameter settings are $r = 0.26$ and $m = (113m - \sqrt{6469m^2 - 6300m})/100$.

Theorem 45.6

There exists an algorithm for PCMT scheduling (MT3) with an approximation ratio

$$r \leq \frac{100}{63} + \frac{100}{345303} \frac{(63m-87)(\sqrt{6469m^2 - 6300m} + 13m)}{m^2 - m} \leq \frac{100}{63} + \frac{100(\sqrt{6469} + 13)}{5481} \approx 3.291919.$$

It is also proved that when m tends to infinity, the optimal asymptotic choices are $\rho = 0.261917$ and $\mu \to 0.325907m$. In this case the ratio $r \to 3.291913$.

Acknowledgment

The authors thank Ulrich Michael Schwarz for helpful comments. Research supported in part by the DFG Graduiertenkollg 357, Effiziente Algorithmen and Mehrskalenmethoden, by the EU Project CRESCCO, Critical Resource Sharing for Cooperation in Complex System, IST-2001-33135, by an MITACS grant of Canada and by the NSERC Discovery Grant DG 5-48923.

References

[1] Culler, D. E., Singh, J. P., and Gupta, A., *Parallel computer architecture: A hardware/software approach*, Morgan Kaufmann, San Francisco, CA, 1999.

[2] Turel, J., Wolf, J., and Yu, P., Approximate algorithms for scheduling parallelizable tasks, *Proc. of SPAA,* 1992, p. 323.

[3] Blayo, E., Debrue, L., Mounié, G., and Trystram, D., Dynamic load balancing for ocean circulation with adaptive meshing, *Proc. Euro-Par*, LNCS 1685, Springer, Berlin, 1999, p. 303.

[4] Lepère, R., Mounié, G., Robič, B., and Trystram, D., Malleable tasks: an electromagnetic efficient model for solving actual parallel applications, *Proc. ParCo* 1999, p. 598.

[5] Brent, P., *The Parallel Evaluation of Arithmetic Expressions in Logarithmic Time*, Academic Press, New York, 1973.

[6] Prasanna, G. N. S. and Musicus, B. R., Generalised multiprocessor scheduling using optimal control, *Proc. of SPAA,* 1991, p. 216.

[7] Prasanna, G. N. S. and Musicus, B. R., Generalized multiprocessor scheduling for directed acyclic graphs, *Proc. of Supercomputing,* 1994, p. 237.

[8] Prasanna, G. N. S. and Musicus, R., The optimal control approach to generalized multiprocessor scheduling, *Algorithmica*, 15(1), 17, 1996.

[9] Garey, M. R. and Johnson, D. S., *Computer and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*, W. H. Freeman, New York, 1979.

[10] Du, J. and Leung, J., Complexity of scheduling parallel task systems, *SIAM J. Discrete Math.*, 2, 473, 1989.

[11] Lenstra, J. K. and Rinnooy Kan, A. H. G., Complexity of scheduling under precedence constraints, *Oper. Res.*, 26, 22, 1978.

[12] Baker, R., Coffman, E. G., and Rivest, R. L., Orthogonal packing in two dimensions, *SIAM J. Comput.*, 9(4), 846, 1980.

[13] Coffman, E. G., Garey, M. R., Johnson, D. S., and Tarjan, R. E., Performance bounds for level-oriented two-dimensional packing algorithms, *SIAM J. Comput.*, 9(4), 808, 1980.

[14] Steinberg, A., A strip-packing algorithm with absolute performance bound 2, *SIAM J. Comput.*, 26(2), 401, 1997.

[15] Kenyon, C. and Rémila, E., A near-optimal solution to a two-dimensional cutting stock problem, *Math. Oper. Res.*, 25(4), 645, 2000.

[16] W. Ludwig and P. Tiwari, Scheduling malleable and nonmalleable parallel tasks, *Proc. of SODA,* 1994, p. 167.

[17] Ludwig, W., Algorithms for Scheduling Malleable and Nonmalleable Parallel Tasks, Ph.D. thesis, University of Wisconsin, Madison, 1995.

[18] Mounié, G., Rapine, C., and Trystram, D., Efficient approximation algorithms for scheduling malleable tasks, *Proc. of SPAA,* 1999, p. 23.

[19] Mounié, G., Rapine, C., and Trystram, D., A 3/2-dual approximation algorithm for scheduling independent monotonic malleable tasks, manuscript.

[20] Jansen, K. and Porkolab, L., Linear-time approximation schemes for scheduling malleable parallel tasks, *Algorithmica*, 32, 507, 2002.

[21] Jansen, K., Scheduling malleable parallel tasks: an asymptotic fully polynomial-time approximation scheme, *Algorithmica*, 39, 59, 2004.

[22] Lepère, R., Mounié, G., and Trystram, D., An approximation algorithm for scheduling trees of malleable tasks, *Eur. J. Oper. Res.*, 142(2), 242, 2002.

[23] Lepère, R., Trystram, D., and Woeginger, G. J., Approximation algorithms for scheduling malleable tasks under precedence constraints, *IJFCS,* 13(4), 613, 2002.

[24] Skutella, M., Approximation algorithms for the discrete time-cost tradeoff problem, *Math. Oper. Res.*, 23 909, 1998.

[25] Zhang, H., Approximation Algorithms for Min–Max Resource Sharing and Malleable Tasks Scheduling, Ph.D. thesis, University of Kiel, Germany, 2004.

[26] Jansen, K. and Zhang, H., Scheduling malleable tasks with precedence constraints, *Proc. of SPAA*, 2005, p. 86.

[27] Jansen, K. and Zhang, H., An approximation algorithm for scheduling malleable tasks under general precedence constraints, *ACM Transactions on Algorithms*, 2(3), 416, 2006.

[28] Graham, R. L., Bounds for certain multiprocessing anomalies, *Bell Syst. Tech. J.*, 45, 1563, 1966.

[29] Martello, S. and Toth, P., *Knapsack problems: Algorithms and computer implementations*, Wiley, New York, 1990.

[30] Papadimitriou, C. H., *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[31] Blazewicz, J., Drabowski, M., and Weglarz, J., Scheduling multiprocessor tasks to minimize schedule length, *IEEE Trans. Comput.*, C-35(5), 389, 1986.

[32] Drozdowski, M., Scheduling multiprocessor tasks—an overview, *Eur. J. Oper. Res.*, 94, 215, 1996.

[33] Jansen, K. and Porkolab, L., Computing optimal preemptive schedules for parallel tasks: linear programming approaches, *Math. Prog.*, 95, 617, 2003.

[34] Grigoriadis, M. D., Khachiyan, L. G., Porkolab, L., and Villavicencio, J., Approximate max–min resource sharing for structured concave optimization, *SIAM J. Optimization*, 11, 1081, 2001.

[35] Lawler, E., Fast approximation algorithms for knapsack problems, *Math. Oper. Res.*, 4, 339, 1979.

[36] Lenstra, J. K., Shmoys, D. B., and Tardos, E., Approximation algorithms for scheduling unrelated parallel machines, *Math. Prog.*, 24, 259, 1990.

[37] Möhring, R. H., Computationally tractable classes of ordered sets, in *Algorithms and Order*, Rival, I., Ed., Kluwer, Dordrecht, 105, 1989.

[38] De, P., Dunne, E. J., Gosh, J. B., and Wells, C. E., The discrete time-cost tradeoff problem revisited, *Eur. J. Oper. Res.*, 81, 225, 1995.

[39] De, P., Dunne, E. J., Gosh, J. B., and Wells, C. E., Complexity of the discrete time-cost tradeoff problem for project networks, *Oper. Res.*, 45, 302, 1997.

46

Vehicle Scheduling Problems in Graphs

Yoshiyuki Karuno
Kyoto Institute of Technology

Hiroshi Nagamochi
Kyoto University

46.1 Introduction

This chapter surveys approximation algorithms for scheduling a set of vehicles that process jobs located at graph vertices. We are given a simple connected graph $G = (V, E)$ with vertex set V, edge set E, and a set J of jobs. The jobs in J (such as items to be picked up or facilities to be inspected) are located at the vertices of the graph. Each job is characterized by a *release time*, a *handling time*, and a *due date* (or a *deadline*). The *time window* for a job is the time interval between its release time and deadline. The handling time for a job means the time required to process the job. The job must be processed without interruption. The handling times are represented by *vertex-weights* in the graph. The *travel time* of an edge of a graph is the time that a vehicle takes to traverse the edge. The travel times are represented by *edge-weights*. Each job must be processed by exactly one vehicle, but any number of vehicles may visit any number of times a vertex without processing its job. A given set of identical vehicles is available to process the set of jobs. All the vehicles process the jobs and traverse the edges at the same speed.

Given p vehicles, the vehicle scheduling problem (VSP) is to find a routing schedule for the p vehicles that minimizes (or maximizes) a given objective function. Objective functions include the maximum tour time of the vehicles, the makespan (i.e., the maximum completion time of jobs), the maximum lateness computed from due dates, and so on. The VSP is an important area of research activity with a variety of industrial and service sector applications [1].

The graphs may be restricted to paths or trees in some applications, for example, ship deliveries along a shoreline [2] and deliveries by robots with elevator boarding functions in a building [3]. The VSP for these graphs has been studied extensively. The VSP restricted to trees is an important subproblem because the bin packing problem is a special case when the tree is a star. Approximation algorithms for bin packing have been extensively studied for the past three decades. Chapter 32 discusses approximation algorithms for bin packing. When the graphs are restricted to trees (resp., paths), we denote the VSP by VSP-TREE (resp., VSP-PATH). The VSP with $p = 1$ is called the *single-vehicle* scheduling problem. We refer to the VSP-TREE (resp., VSP-PATH) with $p = 1$ as the 1-VSP-TREE (resp., 1-VSP-PATH).

Several related routing problems for trees have also been studied such as the p-traveling salesperson problem (p-TSP) and the capacitated vehicle routing problem (CVRP) (see, e.g., Asano et al. [4], Averbakh and Berman [5–7], and Labbé et al. [8]). We do not include a review of approximation algorithms for these related problems, but we do a relationship to the subtree cover problem (SCP). This problem may be a special case of the VSP where each job is characterized only by a *handling time*. Given a simple connected graph $G = (V, E)$, the SCP is find a partition $\mathcal{X} = \{X_1, X_2, \ldots, X_p\}$ of V and a set $\mathcal{T} = \{T_1, T_2, \ldots, T_p\}$ of p subtrees such that each T_i contains X_i so as to minimize (or maximize) a given objective function.

Given an integer p (≥ 2), the *minmax* subtree cover problem (MSCP) is to find a solution $(\mathcal{X}, \mathcal{T})$ that minimizes the maximum cost of the p subtrees, where the cost of each subtree T_i is defined by the sum of edge-weights in T_i and vertex-weights in X_i. When a vertex $v_0 \in V$ is designated as the *root* and each subtree is required to contain v_0, the problem is called the minmax *rooted-subtree* cover problem (MRSCP). As with the VSP, when the graphs are restricted to trees, we use "TREE" that follows the problem name, that is, MSCP-TREE and MRSCP-TREE.

The remainder of this chapter is organized as follows. In Section 46.2, we define graph-theoretical terms and introduce our notation. In Section 46.3, we overview known approximation algorithms for the 1-VSP-TREE and VSP-TREE. We also discuss an important relationship between the VSP and MSCP. Our concluding remarks are given in Section 46.4.

46.2 Preliminaries

Let $G = (V, E)$ be a simple connected graph with a set V of vertices and a set E of edges. Let $n = |V|$ and $m = |E|$. We denote by (G, w) an edge-weighted graph $G = (V, E)$ such that each edge $e \in E$ has a nonnegative weight $w(e)$. The graph is denoted by (G, w, v_0) if a vertex $v_0 \in V$ is designated as the root. The weight $w(e)$ for the edge $e = \{u, v\} \in E$ with end vertices $u, v \in V$ is denoted by $w(u, v)$. The vertex set and edge set of a subgraph $G' \subseteq G$ are denoted by $V(G')$ and $E(G')$, respectively. The sum of the edge-weights in a subgraph $G' \subseteq G$ is denoted by $w(G')$.

Let (J, h) be a subset $J \subseteq V$ of weighted vertices such that each vertex $v \in J$ has a nonnegative weight $h(v)$, and $h(v) = 0$ for every vertex $v \in V \setminus J$. The sum of vertex-weights in a subset $J' \subseteq J$ is denoted by $h(J')$.

A tree is a connected graph containing no cycle. A vertex with degree one is called a *leaf* in a tree, except when it is the root. For a given tree T, a connected subgraph $T' \subseteq T$ is called a *subtree* of T. The set of leaves in T' is denoted by $L(T')$. We define by $b = |L(T)|$ as the number of leaves in tree T. For a tree T and a subset $X \subseteq V(T)$ of vertices, let $T\langle X\rangle$ denote the minimal subtree of T that contains X (where the leaves of $T\langle X\rangle$ will be vertices in X). We say that $T\langle X\rangle$ is *induced* by X (from T).

A path is a tree with exactly two vertices of degree one, called *end vertices*. The other vertices of degree two are called *interior vertices*. A connected subgraph Q' of a path Q is called a *subpath* of Q. In this chapter, a path is referred to as an *end-rooted* path if the root is an end vertex of the path. However, a path is called an *interior-rooted* path if its root is an interior vertex.

We denote by $d_G(u, v)$ the sum of the edge-weights of a shortest path between vertices u and v in a given graph G.

46.3 Vehicle Scheduling Problem

For a graph $(G = (V, E), w)$, the symmetric edge-weight $w(u, v)$ $(= w(v, u))$ for each edge $\{u, v\} \in E$ is the travel time for a vehicle between the two vertices u and v. The vertex set V also represents the job set J. Thus, (V, h) denotes a set of n jobs such that each job $v \in V$ has the handling time $h(v)$. Interruptions are not allowed when processing a job. The set (V, h) is augmented to (V, r, h) if each job v has a release time $r(v)$, that is, the time at which job v becomes available for processing.

For an edge-weighted tree (T, w) and a job set $(V(T), r, h)$, we define the sum of travel times by

$$W = 2w(T) = 2 \sum_{e \in E(T)} w(e)$$

the sum of handling times by

$$H = h(V(T)) = \sum_{v \in V(T)} h(v)$$

and the maximum of release times by

$$r_{max} = \max\{r(v) \mid v \in V(T)\}$$

For a rooted tree (T, w, v_0), we define by

$$d_{max} = \max\{d_T(v_0, v) \mid v \in V(T)\}$$

the travel time from the root v_0 to the farthest vertex from v_0 in T.

The 1-VSP-TREE and VSP-TREE have a large number of variants that depend on time constraints and objective functions. Hereafter, unless otherwise stated, we focus on the following 1-VSP-TREE and VSP-TREE, and provide an overview based on the paper [3].

1-VSP-TREE (Tour time 1-VSP-TREE)

- *Input.* An instance $I = (T, w, v_0, V(T), r, h)$ which consists of a rooted tree (T, w, v_0) and a job set $(V(T), r, h)$.
- *Feasible solution.* A schedule π for a single vehicle initially located at the root v_0, that is, a processing ordering of the n jobs by the vehicle.
- *Goal.* Minimize the tour time $C_{tour}(\pi)$ of the vehicle (i.e., the time at which the vehicle returns to the root v_0 after processing all the jobs).
- *Comments.* In the makespan 1-VSP-TREE, the initial location of the vehicle is also the root v_0, but the vehicle does not have to return to the root.

VSP-TREE (Makespan VSP-TREE)

- *Input.* An instance $I = (T, w, V(T), r, h, p)$ which consists of a tree (T, w), a job set $(V(T), r, h)$, and p identical vehicles $(p \geq 2)$.
- *Feasible solution.* A schedule π for the p vehicles, that is, a set of p (processing) orderings of jobs where each job belongs to exactly one of the p orderings.
- *Goal.* Minimize the makespan $C_{max}(\pi)$ (i.e., the maximum completion time of all jobs).
- *Comments.* You decide the initial locations for the p vehicles, and the vehicles do not have to return to their initial locations. In the maximum tour time VSP-TREE, you decide the initial locations for the vehicles (e.g., choose them so as to minimize the maximum tour time), and each vehicle must return to its initial location.

Table 46.1 summarizes the approximation ratios known for the 1-VSP-TREE and VSP-TREE.

46.3.1 Single-Vehicle Scheduling

In this subsection, we review the known approximation algorithms for the 1-VSP-TREE. We also discuss approximation algorithms for the 1-VSP-PATH, which is a restricted version of the 1-VSP-TREE.

46.3.1.1 In Trees

Nagamochi et al. [9] proved the strong NP-hardness of the 1-VSP-TREE even if all handling times are zero.

We will discuss an approximation algorithm for the 1-VSP-TREE derived from the following polynomially solvable restriction on the 1-VSP-TREE. The depth-first routing constraint for the 1-VSP-TREE is defined as follows. Once the single vehicle reaches a vertex v from its parent in the rooted tree (T, w, v_0), it cannot return to the parent unless it has completed all jobs in the subtrees rooted at v. Under this constraint, each edge $\{u, v\} \in E(T)$ is traversed exactly twice (i.e., once from u to v and the other from v to u). We refer to a schedule under this routing constraint as a *depth-first schedule.*

TABLE 46.1 Approximation Bounds to the VSP in Trees

Objective	Tour Time		Makespan	
	PATH	TREE	PATH	TREE
1-VSP-	$\frac{5}{3}$; $O(n^2)$ [15]	2; $O(n)$ [10]	PTAS [13] 2; $O(n)$ [10] 2; $O(n^2)$ [2]	PTAS[a] [13] 2; $O(n)$ [10]
Objective	Maximum Tour Time		Makespan	
VSP-	$\left(3 - \frac{2}{p+1}\right)$; $O(p^2n)$ [3]	$\left(3 - \frac{2}{p+1}\right)$; $O(p^2n)$ [3]	PTAS[b] [18] 2; $O(pn^2)$ [17]	PTAS[a,b] [18] $\left(5 - \frac{4}{p+1}\right)$; $O(p^2n)$ [3]

Note: PTAS—polynomial-time approximation scheme.

[a] For a constant number b of leaves in a given tree.

[b] For a constant number p of vehicles.

Depth-first schedules have the following properties. The sum of handling times is constant in any schedule (i.e., it is equal to H). The sum of travel times is also constant in any depth-first schedule (i.e., it is equal to W). Thus, minimizing the tour time is equivalent to minimizing the sum of idle times under the depth-first routing constraint.

Karuno et al. [10] observed that an optimal depth-first schedule (i.e., a depth-first schedule with the minimum tour time among all depth-first schedules) can be constructed in $O(n \log n)$ time. The $O(n \log n)$ algorithm works as follows. Assume that the vehicle reaches vertex v from its parent at time t and there exist subtrees rooted at v. For each subtree rooted at v, compute the total idle time incurred by the vehicle which starts from v at time t, visits all jobs in the subtree for processing (of course, in the depth-first manner) and returns to v. The algorithm traverses the subtrees one by one in nondecreasing order of the total idle times. The algorithm performs this computation recursively.

Karuno et al. [10] showed in the same paper that any depth-first schedule π^{DF} is a 2-approximation algorithm for the 1-VSP-TREE. An immediate lower bound on the optimal tour time C^*_{tour} is $\max\{r_{max}, W+H\}$. It is not difficult to see that the tour time of π^{DF} satisfies $C_{tour}(\pi^{DF}) \leq r_{max} + W + H$, since the vehicle can traverse the tree processing all jobs without idle time if it starts at time r_{max}. This implies that π^{DF} is a 2-approximation algorithm for the 1-VSP-TREE. A depth-first schedule π^{DF} can be constructed in $O(n)$ time, and thus the 1-VSP-TREE is 2-approximable in $O(n)$ time. Even for an optimal depth-first schedule, this approximation ratio is asymptotically tight [10].

It is important to note that the minimum travel time is guaranteed if and only if the vehicle uses a depth-first schedule. Minimization of the total travel time is important in practice (e.g., to reduce the battery power consumption of the vehicle). So depth-first schedules are useful for more than their approximation value. For the 1-VSP-TREE, Nagamochi et al. [11] showed that once an optimal depth-first schedule with respect to a specified initial vertex has been obtained, minimum tour times for all other initial vertices can be simultaneously computed in $O(n)$ time. Karuno et al. [12] considered a different variant of 1-VSP-TREE where each job has its own due date, but all jobs are available at time zero. The objective is to minimize the maximum lateness with respect to the due dates. They showed that an optimal depth-first schedule with minimum maximum lateness 1-VSP-TREE can be obtained in $O(n \log n)$ time.

For the makespan 1-VSP-TREE the vehicle does not have to return to the root v_0 (or, no cost is incurred to return from the last vertex to the root). An immediate lower bound in this case for the optimal makespan C^*_{max} is $\max\{r_{max}, W+H-d_{max}\}$, where d_{max} denotes the travel time between the root v_0 and the farthest vertex v_{far} in (T, w, v_0). Since a depth-first schedule where the last job is v_{far} can be constructed in $O(n)$ time, the makespan 1-VSP-TREE is also 2-approximable in $O(n)$ time.

Augustine and Seiden [13] showed that the makespan 1-VSP-TREE with a constant number b of leaves admits a *polynomial-time approximation scheme* (PTAS) (i.e., a family of algorithms $\{A_\varepsilon\}$ such that for any positive real $\varepsilon > 0$, A_ε delivers a schedule with the makespan at most $(1 + \varepsilon)$ times the optimal). Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ be a permutation on $\{1, 2, \ldots, n\}$, where $\pi(i)$ denotes the ith job processed by the vehicle, and let $c_\pi(i)$ be the completion time of the ith job in π. For notational convenience, we define $\pi(0) = v_0$ and $c_\pi(0) = 0$, and let $\pi^{-1}(j)$ be the position of job j in π. A schedule π *eagerly* processes job j if for all i such that job j is located on the unique path from $\pi(i-1)$ to $\pi(i)$, either $\pi^{-1}(j) \le i$ or $r(j) > c_\pi(i-1) + d_T(\pi(i-1), j)$ holds. A schedule is called *eager* if π eagerly processes all jobs. Augustine and Seiden [13] proved that there exists an optimal schedule among eager schedules for the makespan 1-VSP-TREE, and based on this, they showed that an optimal schedule can be constructed in polynomial time when the number of leaves in T and the number of distinct release times are bounded above by a constant. An $O(n)$ time approximation scheme for a fixed ε is derived from these facts. However, the complexity is exponential in $1/\varepsilon$.

46.3.1.2 In Paths

We will discuss a special case of the 1-VSP-TREE where the underlying graph is given by a path Q, that is, the 1-VSP-PATH. The objective is to minimize the tour time C_{tour}. Tsitsiklis [14] proved that the 1-VSP-PATH is NP-hard. However, whether the 1-VSP-PATH is strongly NP-hard remains open problem.

Since a path is a tree where the degree of each vertex is at most two, an optimal depth-first schedule to the 1-VSP-PATH can be obtained in $O(n)$, and it is a 2-approximation algorithm [10].

A schedule $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ is said to be *connected* if any nonempty set $U = \{\pi(i+1), \pi(i+2), \ldots, \pi(n)\} \subseteq V(Q)$ of unprocessed vertices satisfies $U = V(Q\langle U\rangle)$. Psaraftis et al. [2] showed that there is an optimal schedule to the makespan 1-VSP-PATH which is connected if all handling times are zero, and that by using dynamic programming, this subproblem can be solved in $O(n^2)$ time. Subsequently Nagamochi et al. [9] extended the result to an arbitrary graph, and presented an $O(n^2 2^n)$ time algorithm.

Gaur et al. [15] proved that the schedule π_1 constructed by this $O(n^2)$ time dynamic programming procedure has performance guarantee

$$C_{tour}(\pi_1) \le \left(1 + \frac{H}{W + H}\right) C^*_{tour} \tag{46.1}$$

where C^*_{tour} is the optimal tour time. Thus, we can easily see that the schedule π_1 is also a 2-approximation algorithm for the 1-VSP-PATH.

Karuno et al. [16] proved that the 1-VSP-PATH is 3/2-approximable in $O(n)$ time if the given path is end-rooted. The lower bounds

$$C^*_{tour} \ge W + H \tag{46.2}$$

$$C^*_{tour} \ge r(v) + h(v) + d_Q(v, v_0) \quad \text{for any job } v \in V(Q) \tag{46.3}$$

can be easily established. For a real value t with $0 \le t \le r_{max}$, let $H(t)$ (resp., $H'(t)$) be the sum of handling times of all jobs $v \in V$ with $r(v) \ge t$ (resp., $r(v) > t$). They showed that

$$C^*_{tour} \ge t + H(t) \tag{46.4}$$

for any t with $0 \le t \le r_{max}$, and that there always exists a t^* such that $H'(t^*) \le t^* \le H(t^*)$. From these inequalities, they established

$$C^*_{tour} \ge 2t^* \ge 2H'(t^*) \tag{46.5}$$

The *two-phase* algorithm [16] makes the vehicle traverse the path as follows (assuming that the vertex v_0 is the left-end vertex of the path Q and v_{n-1} the right-end vertex): In the forward phase, the vehicle travels from the left-end vertex v_0 to the right-end vertex v_{n-1}, processing all jobs whose release times are at most t^*. In the backward phase, the vehicle returns back from v_{n-1} to v_0, processing all remaining jobs. The approximation ratio of this algorithm is derived as follows. Let π' be the schedule obtained by the two-phase algorithm. If the vehicle does not wait at any vertex in the backward phase, then $C_{tour}(\pi') \le t^* + W + H$.

This is because the vehicle can process all jobs whose release times are at most t^* with no waiting if it waits at the v_0 until time t^*. By Eq. (46.2) and Eq. (46.5), we have

$$C_{tour}(\pi') \leq t^* + W + H \leq t^* + C^*_{tour} \leq \frac{3}{2} C^*_{tour}$$

However, if the vehicle waits at some vertex v_k in the backward phase, then for such a v_k that it is the nearest one to vertex v_0, we have $C_{tour}(\pi') = r(v_k) + h(v_k) + d_Q(v_k, v_0) + H_k$, where H_k denotes the sum of handling times of all jobs that are processed after v_k in π'. Note that $H_k \leq H'(t^*)$ holds. By Eq. (46.3) and again by Eq. (46.5), we obtain

$$C_{tour}(\pi') = r(v_k) + h(v_k) + d_Q(v_k, v_0) + H_k \leq C^*_{tour} + H'(t^*) \leq \frac{3}{2} C^*_{tour}$$

Therefore, the approximation bound of the two-phase algorithm is 3/2.

Gaur et al. [15] modified the two-phase algorithm for the case of interior-rooted paths. In their modified algorithm, the vehicle starts from the root and heads for the closer end to the root. Once the vehicle reaches the closer end, the vehicle follows the two-phase algorithm. Finally, the vehicle returns back from the closer end to the root. Let π_2 be the schedule generated by the modified two-phase algorithm. Then the modified two-phase algorithm has the following performance guarantee for the 1-VSP-PATH [15]:

$$C_{tour}(\pi_2) \leq \left(\frac{3}{2} + \frac{1}{2} \cdot \frac{W}{W + H} \right) C^*_{tour} \qquad (46.6)$$

Gaur et al. [15] showed that the approximation bound 5/3 for the 1-VSP-PATH can be derived from Eq. (46.1) and Eq. (46.6) if one chooses the better from two schedules π_1 and π_2. This is because $1 + H/(W + H) \leq 5/3$ holds if $H \leq 2W$, while $3/2 + W/(2(W + H)) \leq 5/3$ holds if $H > 2W$.

Of course, the PTAS for the 1-VSP-TREE with a constant number b of leaves proposed by Augustine and Seiden [13] can be applied to the 1-VSP-PATH, since $b \leq 2$ holds for paths.

Tsitsiklis [14] considered a different makespan 1-VSP-PATH where each job has its own deadline, but all handling times are zero and all jobs are available at time zero, and showed that an optimal schedule can be computed by an $O(n^2)$ time dynamic programming procedure (which returns $C^*_{max} = \infty$ if no feasible solution exists).

46.3.2 Multi-Vehicle Scheduling

In this subsection, we begin with the VSP-PATH. The first constant factor approximation algorithm to the VSP-PATH was proposed by Karuno and Nagamochi [17]. For a constant number p of vehicles, they also developed a PTAS to the VSP-PATH [18]. Subsequently, a polynomial time constant factor approximation algorithm for the VSP-TREE was derived from a constant factor approximation algorithm for the MSCP-TREE [3].

It should be noted that if all edge-weights in a given path (i.e., travel times of the vehicles) are zero, then the VSP-PATH is identical to the parallel machine scheduling problem, which asks to construct a schedule with minimum makespan under the release time constraint. This machine scheduling problem is denoted by $P/r_j/C_{max}$ under the traditional notation for machine scheduling problems [19].

When $p = 1$, problem $P/r_j/C_{max}$ becomes the single-machine scheduling problem denoted by $1/r_j/C_{max}$. This problem can be solved in $O(n \log n)$ time by scheduling the jobs in nondecreasing order of their release times. However, for the makespan 1-VSP-PATH, a slight change in the processing order of the jobs may affect the makespan dramatically because of the travel times. In fact, Tsitsiklis [14] proved the NP-hardness of the makespan 1-VSP-PATH, which indicates that introducing travel times distinguishes the computational complexity for the makespan 1-VSP-PATH from the $1/r_j/C_{max}$ problem.

Problem $P/r_j/C_{max}$ is NP-hard in the strong sense since it contains the 3-PARTITION as a special case, and the problem for any fixed $p \geq 2$ is NP-hard since it contains the PARTITION (see Garey and Johnson [20]). Hall and Shmoys [21] observed that there exist a 2-approximation algorithm and a PTAS for problem $P/r_j/C_{max}$ (but the algorithm running times are not discussed in their paper). The VSP-PATH is NP-hard in the strong sense for p arbitrary, since it can be viewed as a generalization of $P/r_j/C_{max}$. Similarly, the VSP-PATH is NP-hard even for any fixed $p \geq 2$.

Deriving good lower bounds for the optimal solution value for the VSP-PATH and the VSP-TREE is significantly harder than for the 1-VSP-PATH and the 1-VSP-TREE. In an optimal schedule for an instance of the VSP-PATH with $p \geq 2$, some edges may not be traversed by any vehicle. Such an edge is called a *gap*. This is what makes it difficult to derive a lower bound on the total travel time.

A subpath traversed by a vehicle in a schedule for the VSP-PATH is called its *zone*. A feasible schedule for p' vehicles ($p' \leq p$) is called a *zone schedule* if no two zones intersect and thus there are $p' - 1$ gaps. Moreover, a zone schedule is called a *one-way* zone schedule if any vehicle traverses its zone in one direction only (i.e., from left to right or from right to left). However, a schedule is called *gapless* if each edge is traversed at least once by some vehicle.

Karuno and Nagamochi [17] first observed that there exists a one-way zone schedule which is a 2-approximation algorithm for the optimal gapless VSP-PATH with $p \geq 2$ (i.e., the one with the minimum makespan among all schedules with no gaps). The one-way zone schedule can be constructed in $O(n)$ time. For an optimal gapless schedule, we can establish an immediate lower bound $(W/2 + H)/p$ on its makespan. Notice that a general schedule consists of several gapless schedules for subpaths on a given path. As stated above, each of such subpaths admits a one-way zone schedule that is a 2-approximate solution. Therefore, it suffices to take into account all possible configurations of gaps in the given path. Karuno and Nagamochi [17] proved that an optimal one-way zone schedule can be constructed in $O(pn^2)$ time via dynamic programming. This implies that there exists a one-way zone schedule that is a 2-approximation algorithm for the general case. By designing an approximation algorithm for constructing an optimal one-way zone schedule, Karuno and Nagamochi also presented a nearly linear time $(2 + \varepsilon)$-approximation algorithm for the VSP-PATH for any fixed $\varepsilon > 0$ [17].

Augustine and Seiden [13] extended their PTAS for the makespan 1-VSP-TREE with a constant number b of leaves to a PTAS for the VSP-PATH for finding an optimal zone schedule (note that it may not be an optimal schedule with respect to general schedules). They generalized the dynamic programming procedure provided by Karuno and Nagamochi [17] to compute the gap configuration.

Karuno and Nagamochi [18] developed a PTAS for the VSP-PATH with a constant number p of vehicles. The approximation scheme is based on rounding the release times, and on the fact that any schedule consists of several gapless schedules. Rounding the release times leads to a problem instance with a constant number of distinct release times. The approximation scheme is a two-fold dynamic programming procedure. One is for computing an optimal schedule to the problem with rounded release times, and the other for finding the best schedule to the original problem by combining several gapless schedules over all choices of gaps in the path. The algorithm can be extended to the VSP-TREE and become a PTAS when the number p of vehicles and the number b of leaves in the tree are bounded by a constant [18].

Nagamochi and Okada [22] observed that the VSP-TREE can be regarded as the MSCP-TREE by ignoring its release times, and that the VSP-TREE is approximable within a constant factor by any constant factor approximation algorithm for the MSCP-TREE. Based on this, Karuno and Nagamochi [3] gave an $O(p^2 n)$ time $(5 - 4/(p + 1))$-approximation algorithm for the VSP-TREE. We extend the observation to the VSP in an arbitrary class of graphs.

Theorem 46.1

Let $\mathcal{G}$ be a class of graphs. Assume that there exists an $O(f(n))$ time α-approximation algorithm for the MSCP in an arbitrary graph $G \in \mathcal{G}$. Then a $(1 + 2\alpha)$-approximate solution to the makespan VSP in a graph $G \in \mathcal{G}$ can be obtained in $O(f(n))$ time.

We can obtain an analogous result for the case where the maximum tour time VSP has a root (i.e., a designated vertex to which all vehicles are required to return at the end of a schedule) by using an $O(f(n))$ time α-approximation algorithm for the MRSCP instead of that for the MSCP.

Table 46.2 summarizes the best approximation ratios currently for the MSCP and MRSCP.

TABLE 46.2 Approximation Bounds to the SCP

Underlying Graphs	Trees	Cacti	Arbitrary
MSCP	$\left(2 - \frac{2}{p+1}\right)$ [23]	$\left(4 - \frac{4}{p+1}\right)$ [24]	$(4+\varepsilon)$ [25]
MRSCP	$(2+\varepsilon)$ [23]	$\left(3 - \frac{2}{p+1}\right)$ [26]	$\left(3 - \frac{2}{p+1}\right)$ [26]

46.4 Concluding Remarks

In this chapter, we gave an overview of approximation bounds to the 1-VSP-TREE and VSP-TREE obtained by the previous work. We also provided an important relationship between the VSP and SCP in arbitrary graphs. It is left for the future research to develop algorithms to the SCP that achieve a better performance. In particular, finding a PTAS for the SCP is a challenging problem.

Acknowledgment

This research was partially supported by a Scientific Grant in Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] Desrosiers, J., Dumas, Y., Solomon, M. M., and Soumis, F., Time constrained routing and scheduling, in *Handbooks in Operations Research and Management Science Volume 8: Network Routing*, Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., Eds., North-Holland, Amsterdam, 1995, p. 35.

[2] Psaraftis, H., Solomon, M., Magnanti, T., and Kim, T., Routing and scheduling on a shoreline with release times, *Manage. Sci.*, 36, 212, 1990.

[3] Karuno, Y. and Nagamochi, H., Scheduling vehicles on trees, *Pac. J. Opt.*, 1, 527, 2005.

[4] Asano, T., Katoh, N., and Kawashima, K., A new approximation algorithm for the capacitated vehicle routing problem on a tree, *J. Comb. Optim.*, 5, 213, 2001.

[5] Averbakh, I. and Berman, O., Sales-delivery man problems on treelike networks, *Networks*, 25, 45, 1995.

[6] Averbakh, I. and Berman, O., A heuristic with worst-case analysis for minmax routing of two traveling salesmen on a tree, *Disc. Appl. Math.*, 68, 17, 1996.

[7] Averbakh, I. and Berman, O., $(p-1)/(p+1)$-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective, *Disc. Appl. Math.*, 75, 201, 1997.

[8] Labbé, M., Laporte, G., and Mercure, H., Capacitated vehicle routing on trees, *Oper. Res.*, 39, 616, 1991.

[9] Nagamochi, H., Mochizuki, K., and Ibaraki, T., Complexity of the single vehicle scheduling problem on graphs, *Inf. Sys. Oper. Res.*, 35, 256, 1997.

[10] Karuno, Y., Nagamochi, H., and Ibaraki, T., Vehicle scheduling on a tree with release and handling times, *Ann. Oper. Res.*, 69, 193, 1997.

[11] Nagamochi, H., Mochizuki, K., and Ibaraki, T., Solving the single-vehicle scheduling problems for all home locations under depth-first routing on a tree, *Inst. Electron. Inf. Comm. Eng. Trans. Fundam.*, E84-A, 1135, 2001.

[12] Karuno, Y., Nagamochi, H., and Ibaraki, T., Vehicle scheduling on a tree to minimize maximum lateness, *J. Oper. Res. Soc. Jpn.*, 39, 345, 1996.

[13] Augustine, J. E. and Seiden, S. S., Linear time approximation schemes for vehicle scheduling, in *Algorithm Theory – SWAT*, Lecture Notes in Computer Science, Vol. 2368, Penttonen, M. and Schmidt, E. M., Eds., Springer, Berlin, 2002, p. 30.

[14] Tsitsiklis, J. N., Special cases of traveling salesman and repairman problems with time windows, *Networks*, 22, 263, 1992.

[15] Gaur, D. R., Gupta, A., and Krishnamurti, R., A 5/3-approximation algorithm for scheduling vehicles on a path with release and handling times, *Inf. Proc. Lett.*, 86, 87, 2003.

[16] Karuno, Y., Nagamochi, H., and Ibaraki, T., Better approximation ratios for the single-vehicle scheduling problems on line-shaped networks, *Networks*, 39, 203, 2002.

[17] Karuno, Y. and Nagamochi, H., 2-Approximation algorithms for the multi-vehicle scheduling problem on a path with release and handling times, *Disc. Appl. Math.*, 129, 433, 2003.

[18] Karuno, Y. and Nagamochi, H., An approximability result of the multi-vehicle scheduling problem on a path with release and handling times, *Theor. Comput. Sci.*, 312, 267, 2004.

[19] Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G., Optimization and approximation in deterministic sequencing and scheduling: a survey, *Ann. Disc. Math.*, 5, 287, 1979.

[20] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[21] Hall, L. A. and Shmoys, D. B., Approximation schemes for constrained scheduling problems, *Proc. of FOCS,* 1989, p. 134.

[22] Nagamochi, H. and Okada, K., A faster 2-approximation algorithm for the minmax p-traveling salesmen problem on a tree, *Disc. Appl. Math.*, 140, 103, 2004.

[23] Nagamochi, H. and Okada, K., Polynomial time 2-approximation algorithms for the minmax subtree cover problem, in *Algorithms and Computation – ISAAC*, Lecture Notes in Computer Science, Vol. 2906, Ibaraki, T., Katoh, N., and Ono, H., Eds., Springer, Berlin, 2003, p. 138.

[24] Nagamochi, H. and Kawada, T., Minmax subtree cover problem on cacti, *Disc. Appl. Math.*, 154, 1254, 2006.

[25] Even, G., Garg, N., Könemann, J., Ravi, R., and Sinha, A., Min-max tree covers of graphs, *Oper. Res. Lett.*, 32, 309, 2004.

[26] Nagamochi, H., Approximating the minmax rooted-subtree cover problem, *Inst. Electron. Inf. Comm. Eng. Trans. Fundam.*, E88-A, 1335, 2005.

47

Approximation Algorithms and Heuristics for Classical Planning

Jeremy Frank
NASA Ames Research Center

Ari Jónsson
NASA Ames Research Center

47.1 Introduction

Automated planning has been an active area of research in theoretical computer science and artificial intelligence (AI) for over 40 years. Planning is the study of general purpose algorithms that accept as input an *initial state*, a set of desired *goal states*, and a *planning domain model* that describes how *actions* can transform the state. The problem is to find a sequence of actions that transforms the initial state into one of the goal states. Planning is widely applicable, and has been used in such diverse application domains as spacecraft control [1], planetary rover operations [2], automated nursing aides [3], image processing [4], computer security [5], and automated manufacturing [6]. Planning is also the subject of continued and lively ongoing research.

This chapter will present an overview of how approximations and related techniques are used in automated planning. The chapter focuses on *classical planning* problems, where states are conjunctions of propositions, all state information is known to the planner, and all action outcomes are deterministic. Classical planning is nonetheless a large problem class that generalizes many combinatorial problems including bin packing (Chapters 32–35), prize collecting TSP (Chapter 40) and scheduling (Chapters 44–46). There are numerous planning problems that capture models of uncertainty in the world and in action outcomes; readers interested in learning more about planning are referred to Ref. [7].

47.1.1 Classical Planning

The core of classical planning problems is goal achievement; given a particular state of the world, and actions that can manipulate the state, the problem is to find a sequence of actions that lead to one of the designated set of goal states. More formally, a *planning domain* D consists of a set of world states W and a set of actions A. An action a defines a deterministic mapping $a : W \to W$. A *planning problem instance* $\langle D, i, G\rangle$ consists of a planning domain $D = \langle A, W\rangle$, an initial state $i \in W$, and a set of goal states $G \subset W$.

The set of actions define a directed graph on W; thus, the problem is to find a path from i to a state $g \in G$ or to prove that no such path exists.

When described this way, planning appears quite easy; the problem is either to decide whether there is a path from i to any node g. However, W and the associated set of actions A are usually described *implicitly* by exponentially compressing the descriptions of W and A. The elementary problem description in classical planning uses a formalism called *STRIPS*. The STRIPS formalism [8] uses a set of propositions P to implicitly define the set of world states; $W = 2^P$, with each state $w = \{p_1, \ldots, p_n\}$ being interpreted as the conjunction $(p_1 \wedge \cdots \wedge p_n)$. Each action a is described by a set of preconditions $pre(a)$ (propositions that must be true to enable a), a set of added effects $add(a)$ (propositions that a makes true), and a set of delete effects $del(a)$ (propositions that a makes false). An action a is applicable in a state if the propositions in $pre(a)$ are true in that state. The value of a proposition that does not appear in $add(a)$ or $del(a)$ is unchanged by the application of a.

Given this description of a planning domain, a STRIPS planning problem is defined by a single initial state $i \in 2^P$, and a set of goal propositions $G_p \subseteq P$, implicitly defining a set of goal states $G = \{v \in 2^P \mid G_p \subseteq v\}$. A plan can be viewed as an ordered sequence of actions. A plan can also be viewed as an ordered sequence of concurrent action *sets*; two actions a, b may be *concurrent* in a plan if $(pre(a) \cup add(a)) \cap del(b) = \emptyset$ and $(pre(b) \cup add(b)) \cap del(a) = \emptyset$. A plan is valid if, for each action set, all actions in the set may be concurrent, and if every action in the set is applicable in the state resulting from the sequential application of the previous action sets. The core decision problem is to find a valid plan, that is, a sequence of applicable actions sets $(A_1, \ldots, A_n)$ that maps the initial state to a goal state.

Figure 47.1 shows a simple STRIPS version of a planetary rover domain. The domain consists of a mobile rover with two instruments, a camera and a drill. In this example, the propositions are described using a combination of predicates and parameters; for example, the predicate `at` takes a location parameter. Initially, the rover is at the `Lander`, it has collected no images or samples, and its drill and camera are off. The rover can drive from the `Lander` to a `Hill`, and then to one of the two rocks. The rover can take pictures or sample with the drill, but not both at the same time. The rover cannot drive with the drill on. Finally, the rover's goals are to take a picture at `Rock1`, and both a picture and a drill sample at `Rock2`.

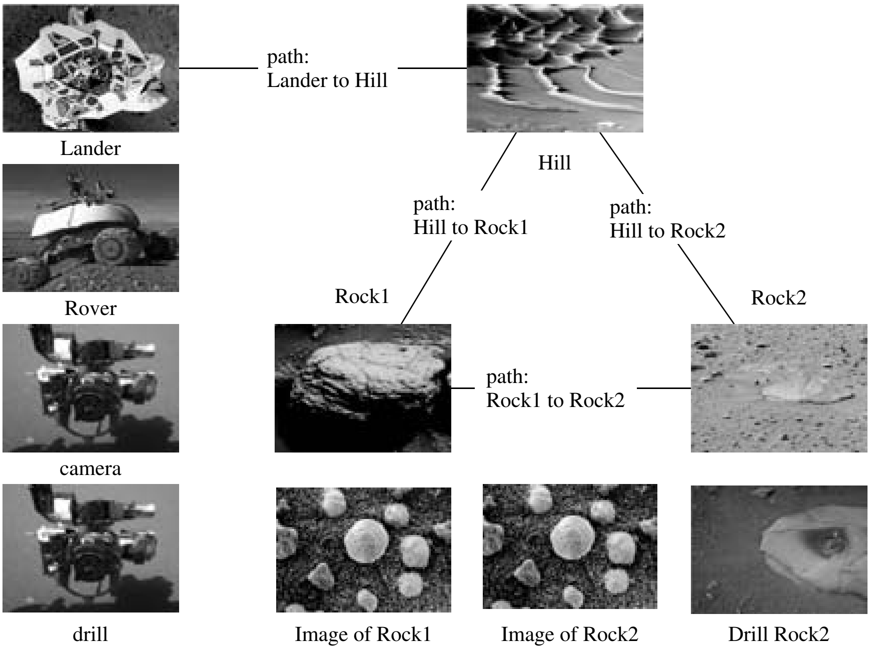
Optimal planning couples the constraints on reaching goals with a function mapping valid plans to values. Common cost functions are the makespan of the plan (i.e., the number of action sets in the plan), the number of actions in the plan, and the utility of a plan. The utility of the plan is the sum of the rewards for propositions in the goal state reached, minus the sum of action costs. Consider again Figure 47.1. Suppose there are two rovers at the lander instead of one; the goal is to gather an image of each rock, and to sample `Rock2`.[1] A minimum makespan plan would have two rovers to go to `Rock2`, since one can drill and the other can take the image. Since the camera can stay on while driving, whichever rover took the image at `Rock2` could drive to `Rock1` and take the image there. By contrast, a minimum action plan would use only one rover, and would first go to `Rock1`, leave the camera on while going from `Rock1` to `Rock2`, then do the drilling operation. The minimum makespan plan has makespan five and uses 10 actions. The minimum action plan, by contrast, has makespan nine but uses only nine actions. It is clear how adding action cost and goal reward to these examples creates complex and interesting problems.

It should be noted that planning research has been moving toward more complex representations of classical planning problems that compactly express constraints involving time, resources, and more complex planning domain rules. These extensions are discussed briefly at the end of the chapter.

47.1.2 Methods for Solving Classical Planning Problems

Planning has motivated considerable work on approximation algorithms, local search techniques, exploitation of structure (e.g., divide-and-conquer methods), and the use of approximations as components of

[1] The STRIPS domain would need to be extended to include action descriptions for the second rover; the extended description is omitted for brevity.



Init:
can-drive(Lander,Hill)
can-drive(Hill, Rock1)
can-drive(Hill, Rock2)
can-drive(Rock1,Rock2)
can-drive(Rock1,Hill)
can-drive(Rock2,Hill)
can-drive(Rock2,Rock1)
no-image(?loc)
no-sample(?loc)
off-drill
off-camera
at(Lander)

Goal:
image(Rock1)
image(Rock2)
sample(Rock2)

action drive(?loc-1, ?loc-2)
pre: at(?loc-1), off-drill,
can-drive(?loc-1,?loc-2)
add: at(?loc-2)
del: at(?loc-1)

action turn-on-camera
pre:off-camera, off-drill
add: ready-camera
del: off-camera

action turn-on-drill(?loc)
pre:off-camera, off-drill, at(?loc)
add: ready-drill
del: off-drill

action drill-sample(?loc):
pre: at(?loc), ready-drill
add: sample(?loc)
delete:no-sample(?loc)

action take-pic(?loc):
pre: at(?loc), ready-camera
add: image(?loc)
delete:no-image(?loc)

action turn-off-camera
pre: on-camera
add:off-camera
delete:on-camera

action turn-off-drill
pre: on-drill
add:off-drill
delete:on-drill

FIGURE 47.1 The STRIPS encoding of a simple planetary rover domain. The `can-drive` propositions describe the simple road network in the picture. The rover is permitted to drive with the camera on, but cannot drive with the drill on. The drill and camera both cannot be turned on simultaneously. The rover must be `at` a location to take a picture of sample.

exact planning algorithms. The motivation behind this is the computational complexity of planning. The complexity is typically measured in the size of the domain, which is usually dominated by the size of the action descriptions A. Standard STRIPS problems, where A is finite, are $\mathcal{PSPACE}$-complete. Restrictions on the problem specification can reduce the computational complexity. For example, the class of planning problems where the length of the plan is bounded is $\mathcal{NP}$-complete. This section gives a broad overview of commonly used search methods to solve planning problems, which will set the stage for a detailed discussion of approximation and local search techniques.

Search strategies for planning can be characterized by the search space representation and algorithm for traversing the search space. One of the most commonly used strategies for planning is *state-space search*, in which the planner searches for a path between the initial state and a goal state. The state space and the set of possible transitions is both implicit and exponential in size. To save time and space, states and allowed state transitions are identified "on the fly" during planning. This leads to two primary state-space search strategies; *progression*, which involves searching from the initial state toward a goal state, and *regression search*, which starts with the propositions in the goal states and searches backward for a path to a state that contains only propositions that are true in the initial state. Another strategy, *plan-space search*, searches the set of possible plans. Typically, a plan-space search algorithm starts with the empty plan as its current candidate plan. At each point in the search, if the candidate plan is not yet a valid solution, the algorithm identifies a flaw that prevents the candidate plan from being a solution, and searches over flaw resolution strategies. A candidate plan may not be valid because a goal has not been established by an action, or a precondition for an action is not guaranteed to be true; one class of resolution strategies is the set of actions that adds the required precondition or goal. Plan-space search is not restricted to progression or regression, but can generate arbitrary partial plans.

Solving planning problems with a systematic search algorithm requires heuristics to guide the search process. In the most general sense, *heuristics* provide guidance on how to make choices in the search process. In state space search, heuristics are evaluated on states or sets of propositions, and estimate the minimal distance to the search objective. For progression, this distance is from the current state to a goal state, while for regression search the distance is from the current set of propositions to a subset of the initial state. *Admissible* heuristics always under-estimate the distance to the goal in state space. Admissible heuristics are appealing because, when used with breadth-first search, the first feasible solution found is also an optimal solution. In state-space search, this yields optimal plans. Unfortunately, there is a tradeoff between admissibility and accuracy, which is manifested in the speed of search. In plan-space search, heuristics are more complex since a plan need not define a single current state that can be compared to the goal. For example, a plan containing actions with missing preconditions implicitly identifies a *set of paths* through the state space, but none of these paths may be valid.

This section discusses a variety of uses of *relaxations* in planning algorithms. One common use of relaxations is to generate efficient, accurate heuristics to guide search; these techniques are discussed in Section 47.2.1. The solution of the simpler problem can be used to guide the search; for example, it can be used as an input to a heuristic choice ranking function, or as an indicator of which choices to pursue and which to ignore. This section will focus on relaxation-based heuristics for state-space search, but will mention some methods used to guide plan-space search. Another use of relaxations is to *approximate* the solutions to planning problems. These methods are described in Section 47.2.2. Approximations are most often used for optimal planning problems, but sometimes are used for decision problems as well. A second class of search guidance techniques relies on identifying and exploiting *problem structure.* These techniques are described in Section 47.2.3. Planning problems often contain *subproblems* that can be solved by specialized combinatorial algorithms, for example, packing, path planning, and scheduling. Another common form of structure exploitation is to *partition* the problem by either ordering or separating goals, thus creating smaller, more easily solved planning problems.

Local search is a powerful technique that has been used successfully on a variety of combinatorial problems. We describe local search approaches to planning in Section 47.3. Planning algorithms employing local search can be applied directly to the space of plans; neighborhood selection for these algorithms can be constructed by using variants of the cost functions discussed in Section 47.2. Local search can also be done by transforming the space of plans into another space that is more amenable to the existing local search algorithms.

47.2 Relaxations of Classical Planning Problems

Approximations are the foundation for many of the search guidance techniques used in planning. The basic idea is to automatically generate an approximation of the given problem, solve or analyze the approximation, and derive useful heuristic information from the result. Relaxations and related approximation techniques are particularly appealing, as they can be relatively easy to solve, while still retaining important aspects of the original problem, and thus being more likely to provide useful information. This section examines relaxation and approximation techniques used to guide planners. The section starts with approximation-based heuristics, then moves on to the use of similar techniques for search space pruning. Finally, the section examines methods that identify subproblems in planning and uses their solutions as heuristics to guide the search for the larger problem.

47.2.1 Heuristic Guidance Using Relaxed Planning Problems

Effective search guidance is essential to the success of systematic search methods for planning. For planning algorithms this means automatically calculating heuristics from the problem specification. The challenge is to make such heuristics accurate enough to guide search effectively while preserving speed.

We will begin by considering heuristic estimation in state-space planning. Since planning goals are stated only in terms of what propositions are true in goal states, the principal source of difficulty in finding plans is the effect of action delete lists. A *relaxed planning problem* is one where all the delete lists have been eliminated. While the existence of a plan can be determined in polynomial time for relaxed planning problems, finding a plan with a minimal number of actions, that is, the path to the nearest goal, is still $\mathcal{NP}$-complete (the minimal action set problem is easily reduced to minimal set covering). Arbitrary solutions to the relaxed planning problem offer poor heuristic guidance, so more refined approximations are needed. One approach, used in progression, is to estimate the distance to individual propositions of a goal state separately, and then combine the estimates. This approach, introduced in the Heuristic Search Planner (HSP) family [9], can be described by a cost function $c(s, p)$ that estimates the number of actions needed to make a proposition p true when starting in state s. The cost estimate, which is easily calculated in low-order polynomial time, is defined by

$$c(s, p) = \begin{cases} 0 & \text{if } p \in s \\ 1 + \min_{a \in A,\, p \in add(a)} f(\{c(s', p) | s' \in pre(a)\}) \end{cases}$$

Letting $f = \mathtt{max}$ when combining costs yields a heuristic $h_{\mathtt{max}}$ that is admissible, whereas letting $f = \sum$ yields a heuristic $h_{\mathtt{add}}$ that may underestimate or overestimate the real cost. A variation on this idea is found in the FastForward (FF) planner [10], but instead of combining estimates for individual propositions, the full relaxed planning problem is solved in a way that biases it toward short solutions. While this does not optimally solve the relaxed problem, the results are often sufficiently accurate to provide a useful heuristic estimate.

Consider our sample domain with the initial state $\mathtt{at(Rock1)} \wedge \mathtt{camera - off}$ and the goal of having $\mathtt{image(Rock2)} \wedge \mathtt{at(Rock1)}$. The shortest plan has four actions, while both $h_{\mathtt{max}}$ and $h_{\mathtt{add}}$ give an estimate of three. It is worth noting that while $h_{\mathtt{max}}$ is admissible, $h_{\mathtt{add}}$ is neither an upper or a lower bound. To see that, consider the same initial state, with the goal of having $\mathtt{image(Rock1)} \wedge \mathtt{image(Rock2)}$. The shortest solution is a four-step plan, while $h_{\mathtt{max}}$ estimates three steps and $h_{\mathtt{add}}$ estimates five steps.

The same idea can be applied in regression search, but with some notable differences. For one, since the distance estimate is always from the initial state to a set of propositions, the heuristic can largely be precalculated for each individual proposition, leaving only a simple combination operation; this idea is discussed in Ref. [9]. However, this simple heuristic will often permit much of the regression search effort to be spent on finding paths from sets of propositions which cannot be reached from the initial state, as they contain two or more propositions that are mutually exclusive. This has led to efforts to identify mutually exclusive sets of propositions and thus prune such states from the search. The notion of a plan-graph [11] has turned out to be one effective approach to calculating mutual exclusions. A *plan-graph* is an approximation of the state space that can be calculated easily and represented in a compact fashion. The plan-graph is constructed as follows. The first level consists of the propositions defining the initial state. The next level consists of all *applicable* actions (an action a is applicable if all propositions in *pre*(a) appear in the previous level, and no two are marked as mutually exclusive). Actions that cannot be concurrent (as described in Section 47.1.1) are marked as being mutually exclusive. The following level consists of all the propositions in the initial level, as well as all effects from actions in the second level. Pairs of propositions that can only be achieved by mutually exclusive actions are marked as such. This process continues until the set of propositions and mutual exclusion annotations does not change between successive levels. An example of a plan-graph for the simple rover domain is shown in Figure 47.2.

The approximation provided by plan-graph is a powerful tool for guiding search methods. In the GraphPlan planner [11], a simple plan-space regression search is used, with no particular emphasis on action selection heuristics. Even so, the mutual exclusion annotations significantly reduce the effort spent on inconsistent states. The plan-graph has also been used successfully in state space search, as it provides a useful approximation of the search space and offers a way to identify impossible states in regression search. Furthermore, the plan-graph has been used to formulate planning problems as constraint satisfaction problems [12,13], and as satisfiability problems [14] (discussed in Section 47.3).

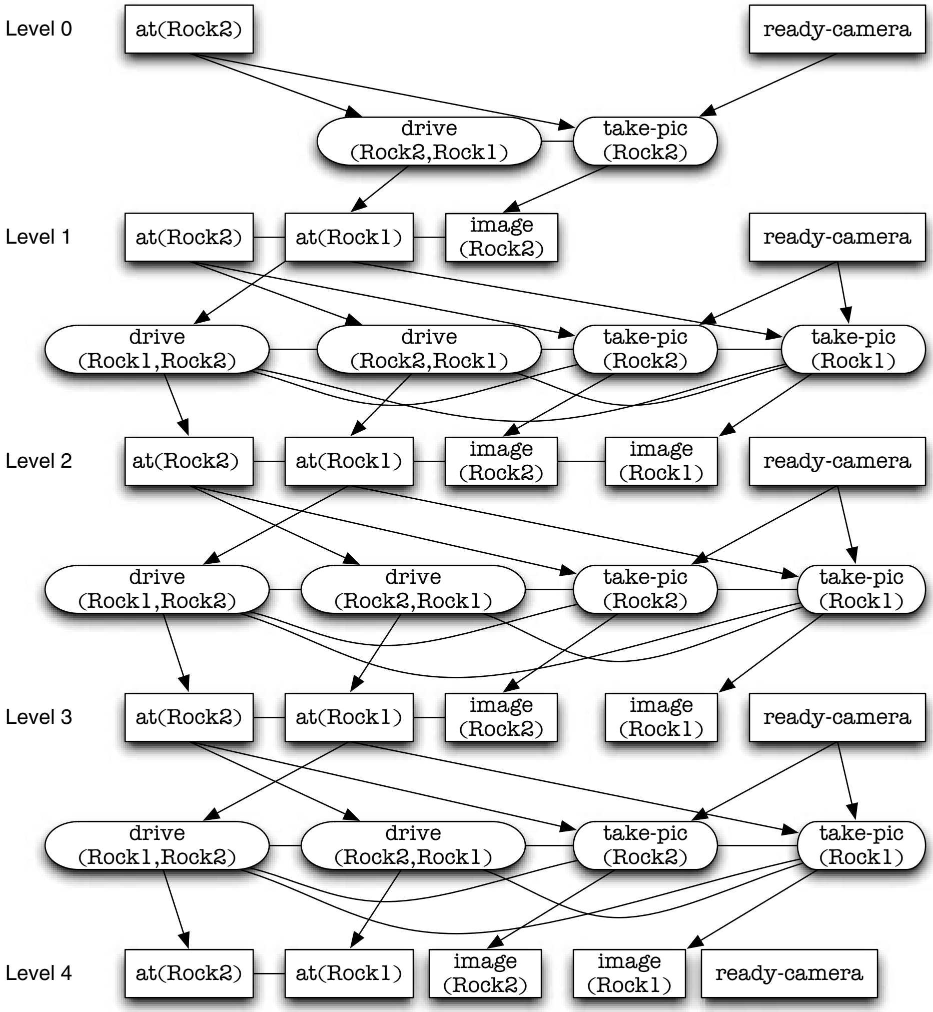


FIGURE 47.2 Part of the plan-graph for the rover domain. Boxes represent propositions and ovals represent actions. Arrows represent action preconditions and effects, while arcs represent mutual exclusions. The plan-graph assumes the rover is `at(Rock2)` and the camera is ready and includes only the actions `drive` and `take-pic` for simplicity. Note that the set of mutual exclusions does not change after level four of this plan-graph.

In many planning algorithms, and especially in state-space search, the plan-graph approximation has been used to generate heuristics. A simple admissible heuristic calculates the distance to a state based on the level of the plan-graph where the propositions in the state first appear without mutual exclusion annotations. For example, in Figure 47.2, the first time `image(Rock1)` and `at(Rock2)` appear together without being mutually exclusive is at level two. The state-space heuristics described above and the plan-graph-based heuristics are in fact related special cases of a more general heuristic [15]. This generalized heuristic is based on calculating the number of actions required to achieve a set of propositions of bounded size b. If $b = 1$, then the derived heuristic is the HSP heuristic; if the $b = 2$, it gives the plan-graph heuristic.

The power of the plan-graph has led to many plan-graph-based heuristics. Simple heuristics often underestimate the number of actions needed to reach a goal, so efforts are typically aimed at improving that. One approach is to partition a set of propositions into minimally interacting subsets and then add up the estimates calculated for each subset. The plan-graph can be used to estimate the interactions between actions and take those into account in heuristic estimates [16].

We note that plan-graph-based heuristics have also been successfully applied to controlling search for certain types of plan-space planners [17]. The idea is to use a special progression planner that focuses search on a limited set of applicable actions. This limitation simplifies the mapping of partial plans to states, making distance estimates easier to calculate.

To this point, the abstractions we have described have been based on relaxing action effects, and using the resulting problems to derive heuristics. Another approach to relaxing planning problems is to eliminate propositions from the problem description. This idea is used in planning with pattern databases [18]. An entry in the database maps a state to a heuristic evaluation that is calculated as the cost of solving a planning problem that is restricted to a subset of the propositions in the planning problem. Such multiple entries, each using a planning problem restricted to a different set of propositions, can be combined into an overall admissible heuristic estimate. The selection of proposition sets is crucial to the value of each entry; typically, the subsets are chosen such that each member has minimal interactions with propositions outside the subset.

Optimal planning problems lead to another set of variations on this theme. These problems require action costs and goal rewards to be folded into the standard plan-graph or state-space search distance estimates. In AltAlt^{PS} and Sapa^{PS} [19], which are progression search and regression planners respectively designed to solve optimal planning problems, the action cost $C(a, t)$ and proposition costs $C(p, t)$ at each level t of the plan-graph are calculated by:

$$C(p, t) = \min(C(a, t-1) + \texttt{cost}(a)$$
$$C(a, t) = f(\{C(s', t) : s' \in pre(a)\})$$

where f is either `min` or $\sum$, and the cost values are initialized with $C(p, 0) = 0$ if $p \in i$, $C(p, 0) = \infty$ if $p \notin i$, and $C(a, 0) = \infty$.

Finally, linear programming approximations that relax variable domains from the propositional set $\{0, 1\}$ to the interval $[0, 1]$ can be used on planning problems. These approximations can be used to guide the search and to prune irrelevant or suboptimal parts of the search space, as is done in many standard integer programming techniques; examples of this technique are described in Refs. [20,21].

47.2.2 Solution Approximations Using Relaxation

While heuristics have been found to help a great deal when solving planning problems, there are many cases when they are not sufficient for solving a problem effectively. This can stem from the heuristics not providing enough information to steer the search effort, or the fact that a significant portion of the search space must be searched to solve a given problem, such as is the case when finding optimal plans or proving that no solution exists. To address these problems, many have turned to methods that reduce the search by *pruning* parts of the search space, often using approximate methods.

Approximate pruning techniques have been explored for most existing planning formulations, but they vary greatly depending on problem being solved and formulation being used. One common notion is to map the problem to an easier class of problem. In those cases, the approximation primarily involves bounding some aspect of the planning problem, such as the length or number of actions. Another common approach is to discard candidates that are viewed as being unlikely to lead to a solution. For example, in the FF planner [10], one method is to limit action selection to "helpful" actions that achieve a desired goal or subgoal. A variation of this approach is to limit the number of options considered at each point.

Optimal planning is a fertile area for approximations for search space pruning, as guaranteed optimality is a particularly difficult problem, necessitating the use of methods that balance the search effort with the quality of the solution returned. The simplest approximation is for optimal planning problems where the solution value is a function of which goal propositions are achieved. Selecting a subset of goal propositions up front turns the problem into a standard planning problem. For example, in AltAlt^{PS} [19], the set of goals is generated iteratively as follows. For each goal g not yet in a current goal set G', solve the relaxed planning problem for goals $G' \cup g$, biased toward reusing the relaxed plan for achieving G'. Then select the one that adds the most value. A more sophisticated variation of the selection strategy examines subsets of candidate goals that are mutually consistent [22].

Another approximation technique is to solve an easier planning problem first, then modify the solution to meet the more difficult objective. This is particularly prevalent in problems where plan length (makespan) must be minimized. One approach to minimizing makespan is to "parallelize" a plan minimizing the number of actions [23]. Two approximations are at work in this technique; not only is finding a minimal length plan from a given sequential plan $\mathcal{NP}$-complete [24], but also an optimal solution to the sequential planning problem does not necessarily provide the best set of actions for a minimal length parallel plan. This is demonstrated by the planning problem employing two rovers described in Section 47.1. Other approaches include adding constraints to limit the search space [25] or heuristics [10] to influence a search technique that otherwise ignores the desired plan optimization criterion.

47.2.3 Heuristic Guidance via Subproblem Identification

Structure and subproblem identification is another powerful tool for guiding search. One class of techniques involves identifying common combinatorial problems as subcomponents of planning problems. Such problems can then be solved with specialized, efficient solvers. The other class of techniques identifies structures like partitions and ordering restrictions, which are then used to split the planning problem into smaller, nearly independent planning problems that can be solved quickly with general-purpose planners, after which the resulting plans are easily merged.

The best known technique for combinatorial subproblem identification uses *type extraction* [26] to identify sets of related propositions and the relations between them. This is done by building up state-machines whose nodes are propositions, and whose transitions are actions. For example, the characteristics of a transportation domain are a set of location propositions, `at(x)`, and a mobility action `move(x,y)` that enables a change in location. These techniques can identify interesting and nonobvious subproblems; an example in Ref. [26] shows how a wall-painting domain leads to a transportation subproblem due to constraints on what colors walls can be painted; that is, the walls are mobile along a map of colors. Type extraction is also extensible to "dependent" maps (e.g., travelers can only travel places with passports), transported objects (packages carried in trucks), and multiple move propositions (drive and fly operators). Furthermore, the approach can be extended to other subproblem types such as packing or scheduling. A significant disadvantage of these techniques is that humans must write the subproblem descriptions. Further work is needed when multiple subproblems can be extracted from the same parent problem instance.

The notion of identifying state machines within planning problems is a powerful idea. A more general approach to identifying problem structures relies on translating planning problems into a different domain modeling language called SAS+ [27], using a transformation algorithm described in Ref. [28]. In this representation, a planning problem consists of finite domain variables, and actions cause subsets of these variables to change values; each variable can be thought of as a state machine. A SAS+ representation of the rover domain is shown in Figure 47.3. A SAS+ domain's *causal graph* is a directed graph over the variables of the problem; there is an edge from u to v if a precondition on u causes v to change value, or an effect

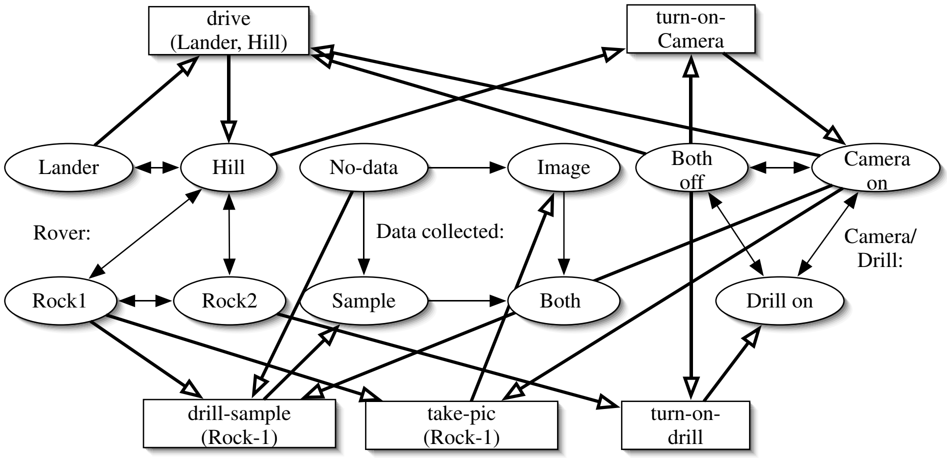


FIGURE 47.3 The rover domain in the SAS+ formalism. Only one instance of each action is provided for simplicity. Ovals are values of state variables, boxes represent actions. State variable value transitions are shown with black arrows, action preconditions and effects are shown with white arrows.

causes both u and v to change value. A subclass of SAS+ problem instances restricts goals to variables with out degree zero in the causal graph. A polynomial-time approximate algorithm for these problems described in Ref. [29] enumerates paths of value transitions from shortest to longest; each path is checked to ensure it satisfies the goal, and that the action descriptions for the low-level goals are satisfied. These plans are converted into distance estimates by summing the number of value transitions for the high-level variable plan and the shortest paths between the initial and final values for the low-level variable.

When it comes to planning problems involving transportation and optimization, a common subproblem is the Orienteering problem or Prize-Collecting TSP (Chapter 40). The Orienteering problem is quite simple compared to an optimal planning problem, consisting of a graph with weighted nodes and edges, where the node weights correspond to edge traversal costs, while node weights correspond to rewards for visiting each edge. Methods for constructing and using the Orienteering problem for optimal planning are described in Ref. [30]. The extraction of the Orienteering problem consists of three phases. The first phase involves a sensitivity analysis step, in which the ground propositions are analyzed for the variable cost of goal achievement. This is done by constructing a relaxed plan for each goal g and calculating costs using the same formula for calculating costs for AltAlt^{PS}. For each proposition p in the final plan for g, assume p was true initially, and that propositions mutually exclusive with p are infinitely expensive, then recalculate the cost. The “seed” of the Orienteering problem comprises the set of all propositions for which the cost of goal achievement varies significantly (using a parameter passed to the algorithm). The problem is complemented by applying each action to each possible state which is both reachable from the initial state, and consists only of propositions in the seed set; this is calculable using a normal plan-graph. The plan-graph is used to estimate the costs of achieving goals at each state of the resulting problem using the same cost function; this process adds new nodes (states derived from the plan-graph) and edges. Approximate solutions to the resulting Orienteering problem then provide heuristics to a classical planner, which can either be an approximate or complete planner.

Another class of structure identification and exploitation techniques relies on identifying subproblems that are solved with a planner instead of special-purpose combinatorial solvers. This approach relies on identifying small subproblems and guaranteeing that an overall solution can be constructed from the subproblems. One method involves finding *landmarks* [31], which are propositions that must hold in all valid plans. These propositions can be goals or intermediate states that must be reached to satisfy the goals. Two landmarks are *necessarily ordered* if one is a precondition for achieving the other. Again referring to Figure 47.1, the goals of imaging `Rock1` and `Rock2` from the initial state of rover at `Lander` both require being at the `Hill` at some point. Deciding if a proposition is a landmark, and deciding whether landmarks are necessarily ordered, are $\mathcal{PSPACE}$-complete problems. A subset of landmarks are found by building the relaxed plan-graph; initially the goals g are landmarks, and subsequently every proposition needed by all actions is a landmark. The relaxed plan-graph is then used to find a subset of the necessary orders. The resulting identified landmarks and orders are used to provide search control to planners, either by controlling decision making or providing guidance to sequential divide-and-conquer algorithms.

47.3 Planning Using Local Search

Local search is a term used to describe a broad range of strategies for solving combinatorial problems. Local search is appealing because it tends to solve such problems fast, but theoretical analysis of average case performance of such algorithms is difficult, and considerable work is often required to tune these algorithms for best performance. This section describes local search techniques for planning.

47.3.1 Local Search Algorithms

Local search algorithms are characterized by a candidate solution to a problem, a set of operators that modify that solution to produce a *neighborhood* of solutions, and a procedure that chooses one of the neighbors as the next candidate solution. When local search is used to solve decision problems, minimizing the number of violated constraints transforms the problem into a pure optimization problem. The most common strategy to solve such problems is a “greedy” one, in which the lowest cost neighbor is selected as the next candidate solution. Greedy local search algorithms can be trapped in *local minima* in which there are no improving moves; numerous strategies for detecting or avoiding local minima have been studied, from injecting random moves and random restarts into search (Chapter 19) to adding either temporary constraints (Chapters 21 and 23) or permanent ones that influence subsequent steps. Another problem is that of *large neighborhoods*, which can be expensive to evaluate. This leads to *early termination* rules that improve the rate at which moves are made, at the possible cost of slow improvement in the optimization criteria (Chapter 20).

Constrained optimization problems can also be transformed into pure optimization problems by assigning a penalty to violated constraints. In constrained optimization problems over continuous variables, these penalties are called Lagrange multipliers (Chapter 4). The theory of Lagrange multipliers states that saddle points in the space of the original variables plus the Lagrange multipliers are solutions to the original constrained optimization problem, as long as some other properties are satisfied; this theory has been extended to discrete problems as well.

47.3.2 Local Search Neighborhoods for Planning

A common method of local search for planning is to define a set of operators that directly manipulate plans, in effect performing local search over plan space. These methods generate a set of possible plans from the current plan. The plans are then assessed to determine whether they are an improvement over the current plan; this assessment drives selection of the next plan in the neighborhood.

When feasible plans are easy to find and optimization is required, local search over the space of plans is a sensible approach. Planning by rewriting or PBR [32] is one such technique; it operates in plan space. The *plan rewrite rules* are domain-specific operators that are designed to create better plans. Rules come in four classes: action reordering rules, collapsing rules (that reduce the number of actions), expanders (inverse of collapsers), and parallelizers (that eliminate ordering constraints). A sample reordering rule from the rover problem (Figure 47.1) is: if the plan moves the rover to accomplish a goal, then moves back to the current position to accomplish another goal, reorder these goals and discard the extra movements. If feasible plans are needed, the violated rules can be assigned penalties. This approach is used in Ref. [33]; specifically, an elaboration on the landmark technique described in Section 47.2 is used to divide the plan into segments. The rules of the domain ensure that the plan is well formed, that is, each segment is well formed, and the actions in segment i establish the preconditions for states in segment $i + 1$.

Another option is to operate on the space of plan-graphs. An action-graph is a subset of a plan-graph such that if an action is in action-graph then the propositions in the precondition and add effect lists are also in action-graph. Action-graphs can have inconsistencies, either in the form of mutual exclusions or propositions not supported by actions. This leads to a natural local search neighborhood; actions can be added to establish support, or removed to eliminate unsupported actions or resolve mutual exclusions. A linear-action-graph is an action-graph with at most one action per level. Local search on linear-action-graphs only manipulates actions to fix unsupported precondition inconsistencies; this leads to a simpler and smaller search neighborhood. Actions may be added at any level; the resulting linear-action-graphis grown by one level. Actions with unsupported preconditions may also be removed. LPG [34] is a local search planner that operates on action-graphs. In Figure 47.4 we show a linear action graph neighborhood for the rover example. In the scenario in Figure 47.4 there is only one way to legally insert the `turn-on-drill` action. In general, however, there may be many neighbors to evaluate. Due to the expense of evaluating the neighborhood, and its potential size, small subsets of inconsistencies may be used to generate neighbors, with heuristics guiding inconsistency selection. Further limits can also be imposed to reduce the neighborhood size.

Another method involves *transforming* the plan space into another representation and performing local search in this representation. One possible representation is the Propositional Satisfiability problem (SAT); considerable work on local search (and complete search) has been done on this problem. Propositions encode assertions such as "plan proposition holds at step" or "action occurs at step," and clauses encode the domain rules governing legal action occurrences and their implications. The transformation of the plan space into SAT requires searching over a restricted set of plans; this is often done in a manner similar to construction of the plan-graph. Different encodings impact the size of the SAT problem and the local search topology, and therefore solver performance. One encoding, described in Ref. [35], works as follows. Actions imply their preconditions at the current instant and their effects at the next instant; one

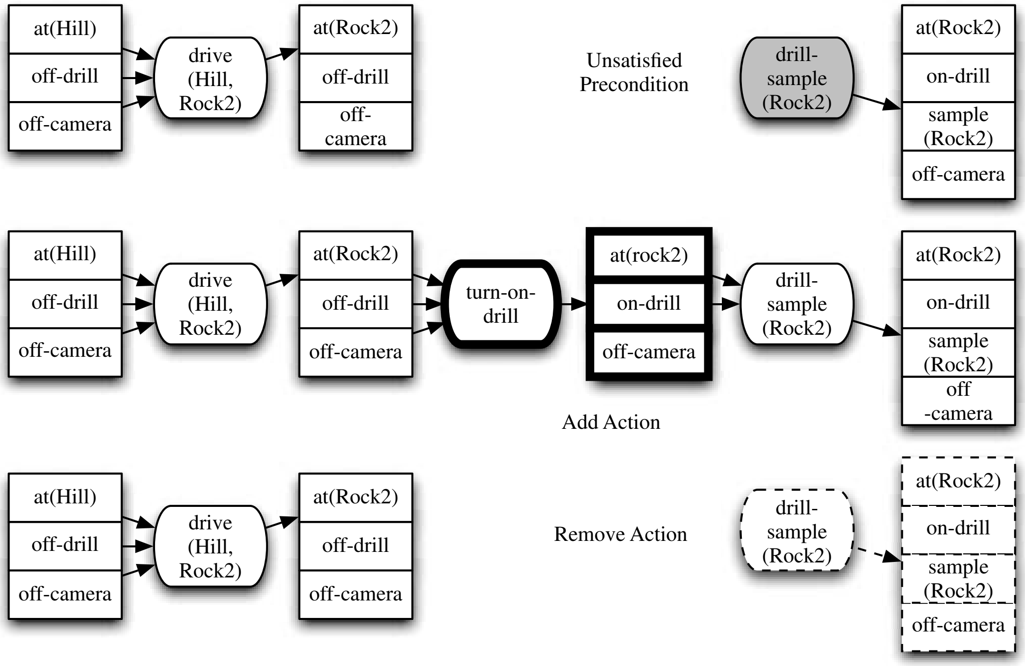


FIGURE 47.4 An example of the search neighborhood for LPG in the rover domain using linear action graphs. The preconditions for `drill-sample` are unsatisfied, inducing a search neighborhood of size two. There is one legal place for adding `turn-on-drill`; alternately, `drill-sample` can be removed.

```
Actions imply preconditions
and effects

(drive-Hill-Rock1-i) =>
((at-Hill-i) ^
(can-drive-Hill-Rock1-i) ^
(at-Rock1-i+1) ^
(¬at-Hill-i+1))

(turn-on-Camera-i) =>
((off-Camera-i) ^
(off-Drill-i) ^
(ready-Camera-i+1) ^
(¬off-Camera-i+1))

(take-pic-Rock1-i) =>
((at-Rover-Rock1-i) ^
(ready-Camera-i) ^
(image-Rock1-i+1))
...
```

```
Only one action at a time

((drive-Hill-Rock1-i) v
(¬turn-on-Camera-i) v
...
(¬Sample-Rock1-i))
^
((¬drive-Hill-Rock1-i v
(turn-on-Camera-i) v
...
(¬Sample-Rock1-i))
^
((¬drive-Hill-Rock1-i) v
(¬ turn-on-Camera-i) v
...
(Sample-Rock1-i))
```

```
Frame axioms
(summarized)

((take-pic-Rock-i) ^
(true-prop-i)) =>
(true-prop-i+1)

((drive-Hill-Rock1-i) ^
true-prop-i)) =>
(true-prop-i+1)

((turn-on-Camera-i) ^
true-prop-i)) =>
(true-prop-i+1))
...
```

FIGURE 47.5 The SATPlan encoding of the rover problem.

action occurs per instant; the initial state holds; and the frame conditions hold. An example of this SAT encoding for our rover domain is shown in Figure 47.5. Another representation, described in Ref. [36], allows concurrent actions, and eliminates the action propositions from the encoding; all that remains are logical clauses expressing the possible transformations of propositions truth values after a (set of) action invocations. This reduces the size of the domain encoding and thus improves SAT solver performance. Other techniques to manipulate the SAT encoding include using unit propagation and posting single- and double-literal assertions to generate new clauses.

SAT is a good representation for finding feasible plans, but more expressive representations can provide more flexibility to solvers. One such representation is a special subset of Integer Linear Programs (ILPs) called Over-constrained Integer Programs (OIPs). The OIP consists of two systems of linear inequalities, one set $A_i x \leq b$ defining feasible plans, and another set $C_i x \leq d$ defining optimization criteria. The optimization function is to minimize the nonlinear function $\sum_i C_i x - d \mid C_i x > d$, that is, the "deviation" of the current solution from the optimal solution. The encoding is an extension of the SAT-based encodings described previously; it is easy to transform SAT into a $0-1$ LP. Linear objective functions based on states and actions in the plan are easily encoded as systems of LPs. This representation is used by WalkSAT(OIP) [37].

47.3.3 Selecting the Successor

Local search algorithms that operate directly on the plan-state face the problem of deciding which neighbor to select. When the set of plans in the neighborhood is mostly feasible, as is the assumption behind PBR, the neighborhood can simply be evaluated by the function that determines the value of feasible plans. When the neighborhood consists of infeasible plans, as occurs with LPG, the problem is how to estimate the relative merit of infeasible plans. One solution is to use heuristics that estimate the amount of work needed to fix the problems with this plan. The situation is well summarized when considering heuristics estimating the work needed to find a feasible linear-action-graph. Let *threats(a,C)* be the set of supporting preconditions of actions in C that become unsupported if a is added to C; *nopre(a,C)* be the set of unsupported preconditions of a in C; and *unsup(b,C)* be the set of supporting preconditions of actions in C that become unsupported if b is removed from C. A simple neighborhood evaluation function is

$$E_0^i(a, C) = |\mathit{threats}(a, C)| + |\mathit{nopre}(a, C)|$$
$$E_0^r(b, C) = |\mathit{unsup}(b, C)|$$

where $E_0^i(a, C)$ is used if a is added to C, and $E_0^r(b, C)$ is used if b is removed from C. A more accurate heuristic recursively adds actions to support unsupported facts, using $E_0^i(a, C)$ and $E_0^r(a, C)$ to penalize these actions. The most accurate heuristic extends the linear-action-graph using a regression planner that ignores delete effects of actions; the heuristic then counts actions in the resulting (relaxed) plan and conflicts between the relaxed plan and the current linear-action-graph.

Representations based on either SAT or constraints, such as OIP, induce simple neighborhoods. The common neighborhood in the SAT solvers used in SATPlan [38] is to reverse the assignment of each proposition. For constraint-based representations like OIP, the set of variable assignments satisfying one violated constraint form the search neighborhood. Representations that partition constraints into feasibility constraints and optimality constraints such as OIP enable algorithms that choose between sets of violated constraints according to a predetermined bias.

Since the theory of Lagrange multipliers has been extended to discrete search, it is possible to transform an optimal planning problem into a discrete constrained optimization problem, and implement a search to find extended saddle points in the Lagrangian space of a problem. This is the approach taken in Ref. [33]. Plans are partitioned into contiguous segments, with discrete choices corresponding to actions that can be added to or removed from each segment; this effectively creates many small, linked optimization problems, thereby parallelizing search. The search for a saddle point utilizes numerous methods to control growth of the Lagrangian penalties, as well as the usual methods of neighborhood cost control and randomization to promote exploration. Additional methods are used to adjust segment boundaries during search. This approach can use many different planners to generate moves during search. This method is more general than using OIPs, but still requires careful modeling of the search space.

47.4 Planning Ahead: The Future of Approximations and Planning

Recent work in AI planning has focused on extending the core goal achievement problem to better represent the complexity of real-world problems. This includes handling temporal and resource constraints found in scheduling problems [39], as well as more complex relations between actions and states. The most popular planning domain description language (PDDL), has been extended to provide more direct support for temporally extended state and action, numeric fluents (meant to capture resource states), and planning with knowledge of future state (exogenous events). In addition, other variants of STRIPS have been developed to include explicit resource declarations, resource-impacting actions, and allow resource state-based preconditions, for example, see Ref. [40].

Work is under way to extend existing techniques to handle the additional complexity, and to develop new approaches to represent and reason about plans. This work includes approximation methods, structure identification techniques, and local search, which will continue to play a crucial role in effective automated planning. For search guidance, plan-graph methods have been extended to include temporal information [41] and numerical fluents [42]. Structure identification and exploitation techniques include the use of special-purpose methods for scheduling subproblems [43]. Local search approaches are being utilized for resource-constrained planning [44]. However, a great deal of work still needs to be done to reach the full potential of approximation techniques and local search methods for planning in the future.

Acknowledgments

The authors gratefully acknowledge Sailesh Ramakrishnan, Ronen Brafman and Michael Freed for reviewing our early drafts and providing editorial assistance. This work was funded by the NASA Intelligent Systems Program.

References

[1] Muscettola, N., Nayak, P., Pell, B., and Williams, B., Remote agent: To boldly go where no AI system has gone before, *Artif. Intelligence*, 103(1–2), 5, 1998.

[2] Bresina, J., Jonsson, A., Morris, P., and Rajan, K., Activity planning for the Mars exploration rovers, *Proc. 15th Int. Conf. on Automated Planning and Scheduling*, 2005, p. 40.

[3] McCarthy, C. E. and Pollack, M. E., A plan-based personalized cognitive orthotic, *Proc. 6th Conf. on Artificial Intelligence Planning Systems*, 2002, p. 243.

[4] Golden, K., Pang, W., Nemani, R., and Votava, P., Automating the processing of earth observation data, *Proc. 7th Int. Symp. on Artificial Intelligence, Robotics and Space*, 2003.

[5] Boddy, M., Gohde, J., Haigh, T., and Harp, S., Course of action generation for cyber security using classical planning, *Proc. 15th Int. Conf. on Automated Planning and Scheduling*, 2005, p. 12.

[6] Ruml, W., Do, M. B., and Fromhertz, M., On-line planning and scheduling for high speed manufacturing, *Proc. 15th Int. Conf. on Automated Planning and Scheduling*, 2005, p. 30.

[7] Traverso, P., Ghallab, M., and Nau, D., *Automated Planning: Theory and Practice*, Morgan Kauffman, San Francisco, CA, 2004.

[8] Fikes, R. and Nilsson, N., STRIPS: A new approach to the application of theorem proving to problem solving, *Artif. Intelligence*, 2(3/4), 189, 1971.

[9] Bonet, B. and Geffner, H., Planning as heuristic search, *Artif. Intelligence*, 129(1–2), 5, 2001.

[10] Hoffmann, J. and Nebel, B., The FF planning system: fast plan generation through heuristic search, *J. Artif. Intelligence Res.*, 14, 253, 2001.

[11] Blum, A. and Furst, M., Fast planning through planning graph analysis, *Proc. 14th Int. Joint Conf. on Artificial Intelligence*, 1995, p. 1636.

[12] Lopez, A. and Bacchus, F., Generalizing graphplan by formulating planning as a CSP, *Proc. 18th Int. Joint Conf. on Artificial Intelligence*, 2003, p. 954.

[13] Do, M. B. and Kambhampati, S., Solving planning-graph by compiling it into CSP, *Proc. 5th Int. Conf. on Artificial Intelligence Planning Systems*, 2000, p. 82.

[14] Kautz, H., McAllester, D., and Selman, B., Encoding plans in propositional logic, *Proc. 5th Int. Conf. on the Principle of Knowledge Representation and Reasoning*, 1996, p. 374.

[15] Haslum, P. and Geffner, H., Admissible heuristics for optimal planning, *Proc. 5th Int. Conf. on Artificial Intelligence Planning Systems*, 2000, p. 140.

[16] Nguyen, X., Kambhampati, S., and Sanchez Nigenda, R., Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search, *Artif. Intelligence*, 135(1–2), 73, 2002.

[17] Nguyen, X. and Kambhampati, S., Reviving partial ordering planning, *Proc. 17th Int. Joint Conf. on Artificial Intelligence*, 2001, p. 459.

[18] Edelkamp, S., Symbolic pattern databases in heuristic search planning, *Proc. 6th Int. Conf. on Artificial Intelligence Planning Systems*, 2002, p. 274.

[19] van den Briel, M., Sanchez Nigenda, R., Do, M. B., and Kambhampati, S., Effective approaches for partial satisfaction (oversubscription) planning, *Proc. 19th National Conf. on Artificial Intelligence*, 2004, p. 562.

[20] Bylander, T., A linear programming heuristic for optimal planning, *Proc. 14th National Conf. on Artificial Intelligence*, 1997, p. 694.

[21] Vossen, T., Ball, M., Lotem, A., and Nau, D., On the use of integer programming models in AI planning, *Proc. 16th Int. Joint Conf. on Artificial Intelligence*, 1999, p. 304.

[22] Sanchez Nigenda, R. and Kambhampati, S., Planning graph heuristics for selecting objectives in over-subscription planning problems, *Proc. 15th Int. Conf. on Automated Planning and Scheduling*, 2005, p. 192.

[23] Do, M. B. and Kambhampati, S., Improving temporal flexibility of position constrained metric temporal plans, *Proc. 13th Int. Conf. on Automated Planning and Scheduling*, 2003, p. 42.

[24] Bäckström, C., Computational aspects of reordering plans, *J. Artif. Intelligence Res.*, 9, 99, 1998.

[25] Buttner, M. and Rintanen, J., Improving parallel planning with constraints on the number of actions, *Proc. 15th Int. Conf. on Automated Planning and Scheduling*, 2005, p. 292.

[26] Long, D. and Fox, M., Automatic synthesis and use of generic types in planning, *Proc. 5th Artificial Intelligence Planning Systems*, 2000, p. 196.

[27] Bäckström, C. and Nebel, B., Complexity results for SAS$^+$ planning, *Comput. Intelligence*, 11(4), 625, 1995.

[28] Edelkamp, S. and Helmert, M., Exhibiting knowledge in planning problems to minimize state encoding length, *Proc. 5th Eur. Conf. on Planning*, 1999, p. 135.

[29] Helmert, M., A planning heuristic based on causal graph analysis, *Proc. 14th Int. Conf. on the Automated Planning and Scheduling*, 2004, p. 161.

[30] Smith, D., Choosing objectives in over-subscription planning, *Proc. 14th Int. Conf. on Automated Planning and Scheduling*, 2004, p. 393.

[31] Hoffmann, J., Porteous, J., and Sebastia, L., Ordered landmarks in planning, *J. Artif. Intelligence Res.*, 22, 215, 2004.

[32] Ambite, J. L. and Knoblock, C. A., Planning by rewriting, *J. Artif. Intelligence Res.*, 15, 207, 2001.

[33] Wah, B. and Chen, Y., Subgoal partitioning and global search for solving temporal planning problems in mixed space, *Int. J. Artif. Intelligence Tools*, 13(4), 767, 2004.

[34] Gerevini, A., Saetti, L., and Serina, I., Planning through stochastic local search and temporal action graphs, *J. Artif. Intelligence Res.*, 20, 239, 2003.

[35] Kautz, H. and Selman, B., Planning as satisfiability, *Proc. 10th Eur. Conf. on Artificial Intelligence*, 1992, p. 359.

[36] Kautz, H. and Selman, B., Unifying sat-based and graph-based planning, *Proc. 16th Int. Joint Conf. on Artificial Intelligence*, 1999, p. 318.

[37] Kautz, H. and Walser, J. P., State-space planning by integer optimization, *Proc. 16th National Conf. on Artificial Intelligence*, 1999, p. 526.

[38] Kautz, H. and Selman, B., Pushing the envelope: Planning, propositional logic and stochastic search, *Proc. 13th National Conf. on Artificial Intelligence*, 1996, p. 1194.

[39] Smith, D., Frank, J., and Jónsson, A., Bridging the gap between planning and scheduling, *Knowledge Eng. Rev.*, 15(1), 2000.

[40] Koehler, J., Planning under resource constraints, *Proc. 13th Eur. Conf. on Artificial Intelligence*, 1998, p. 489.

[41] Smith, D. and Weld, D., Temporal planning with mutual exclusion reasoning, *Proc. 16th Int. Joint Conf. on Artificial Intelligence*, 1999, p. 326.

[42] Hoffmann, J., The metric-FF planning system: Translating "ignoring delete lists" to numeric state variables, *J. Artif. Intelligence Res.*, 20, 291, 2003.

[43] Srivastava, B., Realplan: decoupling causal and resource reasoning in planning, *Proc. 17th National Conf. on Artificial Intelligence*, 2000, p. 812.

[44] Tran, D., Chien, S., Sherwood, R., Castaño, R., Cichy, B., Davies, A., and Rabbideau, G., The autonomous sciencecraft experiment onboard the eo-1 spacecraft, *Proc. 19th National Conf. on Artificial Intelligence*, 2004, p. 1040.

48 Generalized Assignment Problem

Mutsunori Yagiura
Nagoya University

Toshihide Ibaraki
Kwansei Gakuin University

48.1 Introduction

The *generalized assignment problem* (GAP) is one of the representative combinatorial optimization problems known to be NP-hard. Given n jobs and m agents, we undertake to determine a minimum-cost assignment such that every job is assigned to exactly one agent and the resource constraint for each agent is satisfied. This problem is a natural generalization of such representative combinatorial optimization problems as bipartite matching, knapsack, and bin packing problems, and has many important applications, including flexible manufacturing systems [1], facility location [2], and vehicle routing problems [3]. Consequently, designing good, exact, or heuristic algorithms for GAP has significant practical as well as theoretical value. Among various heuristic algorithms developed for GAP are a combination of the greedy method and the local search by Martello and Toth [4,5]; a tabu search and simulated annealing by Osman [6]; a genetic algorithm by Chu and Beasley [7]; variable depth search algorithms by Racer and Amini [8,9]; a tabu search based on ejection chain approach by Laguna et al. [10] (which is proposed for a generalization of GAP); another tabu search by Díaz and Fernández [11]; a Lagrangian heuristic algorithm by Haddadi and Ouzia [12]; a MAX-MIN ant system combined with local search and tabu search by Lourenço and Serra [13]; a path-relinking algorithm by Alfandari et al. [14]; ejection chain approaches by Yagiura et al. [15,16], and so on. Research for exact algorithms also has long history since early papers by Ross and Soland [17], Martello and Toth [4], and Fisher et al. [18]. Among recent exact algorithms successful for GAP are the branch-and-bound methods by Savelsbergh [19], Nauss [20], and Haddadi and Ouzia [21].

In this chapter, we review heuristic and metaheuristic algorithms for GAP. We first define the problem formally, and introduce some related problems and their complexities in Section 48.2. We then explain basic strategies of greedy method and local search in Sections 48.3 and 48.4. As relaxation problems often provide useful information to improve the search, we introduce Lagrangian relaxation problems in

Section 48.5, and then explain some basic ideas of Lagrangian heuristics in Section 48.6. Although branch-and-bound is usually used to design exact algorithms, it can also be used for approximation algorithms; hence we describe the basics of branch-and-bound in Section 48.8. We then report some computational results of various metaheuristic algorithms and branch-and-bound methods in Section 48.9. A brief survey on polynomial-time approximation algorithms with performance guarantees is given in Section 48.10.

48.2 Generalized Assignment Problem

In this section, we first give a formal definition of the generalized assignment problem, and then introduce some equivalent formulations, special cases, and generalizations, and discuss their complexities.

48.2.1 Definition of the Problem

Given n jobs $J = \{1, 2, \ldots, n\}$ and m agents $I = \{1, 2, \ldots, m\}$, we determine a minimum-cost assignment subject to the constraints of assigning each job to exactly one agent and satisfying a resource constraint for each agent. Assigning job j to agent i incurs a cost of c_{ij} and consumes an amount a_{ij} (> 0) of the resource, whereas the total amount of the resource available at agent i is b_i (> 0). An assignment is a mapping σ: $J \to I$, where $\sigma(j) = i$ means that job j is assigned to agent i. For convenience, we define a 0-1 variable x_{ij} for each pair of $i \in I$ and $j \in J$ by

$$x_{ij} = 1 \iff \sigma(j) = i$$

Then the *generalized assignment problem* (GAP) is formulated as follows:

$$\begin{aligned}
\text{GAP:} \qquad & \text{minimize} \qquad && cost(x) = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} && (48.1)\\
& \text{subject to} && \sum_{j \in J} a_{ij} x_{ij} \le b_i, \qquad \forall i \in I && (48.2)\\
& && \sum_{i \in I} x_{ij} = 1, \qquad \forall j \in J && (48.3)\\
& && x_{ij} \in \{0, 1\}, \qquad \forall i \in I \text{ and } \forall j \in J && (48.4)
\end{aligned}$$

The first condition (48.2) signifies that the amount of resource required by the jobs assigned to each agent must not exceed the available amount of resource at the agent, which is called the resource constraint, and the second condition (48.3) signifies that each job must be assigned to exactly one agent, which is called the assignment constraint. The objective value $cost(x)$ is also referred to as $cost(\sigma)$, as it represents the cost of assignment $\sigma : J \to I$.

48.2.2 Related Problems and Complexity

GAP is sometimes defined as a maximization problem whose objective is to maximize $\sum_{i \in I} \sum_{j \in J} \rho_{ij} x_{ij}$, where ρ_{ij} are profits of assigning job j to agent i, subject to constraints (48.2)–(48.4). This is a natural generalization of the knapsack problem [5], and is equivalent to the above minimization formulation; simply let $c_{ij} := -\rho_{ij}$, or let $c_{ij} := \bar{\rho} - \rho_{ij}$ for a constant $\bar{\rho}$ with $\bar{\rho} \ge \max_{i \in I,\, j \in J} \rho_{ij}$ if negative costs are not preferable. In the latter case, $cost(x) = \sum_{i \in I} \sum_{j \in J} (\bar{\rho} - \rho_{ij}) x_{ij} = n\bar{\rho} - \sum_{i \in I} \sum_{j \in J} \rho_{ij} x_{ij}$ holds for any feasible x by (48.3).

GAP is known to be NP-hard, and the (supposedly) simpler problem of determining the existence of a feasible solution for GAP is NP-complete in the strong sense, since (the decision version of) the bin packing problem [22] can be reduced to this problem. This feasibility problem of GAP remains to be NP-complete even if m is a fixed constant, since the partition problem [22] can be reduced to this problem with $m = 2$.

In contrast, we can consider the problem GAP′, which is GAP with constraint (48.3) being replaced by

$$\sum_{i\in I} x_{ij} \leq 1, \quad \forall j \in J \tag{48.5}$$

This GAP′ always has a feasible solution. Note that in this case, a job j will not be assigned to an agent i unless $c_{ij} \leq 0$, and hence the formulation of profit maximization seems more natural. Such maximization formulation is suitable when performance guarantees are considered [23,24], and is called by various names such as LLGAP [5] and max-profit GAP [24].) GAP′ is, however, equivalent to GAP as shown below. For any instance of GAP′, we can construct an equivalent instance of GAP by adding a dummy agent "$m+1$" with $b_{m+1} = n$, $c_{m+1,j} = 0$, and $a_{m+1,j} = 1$ for all $j \in J$. That is, corresponding to a solution of GAP′ with $\sum_{i\in I} x_{ij} = 0$, GAP has the solution with $x_{m+1,\,j} = 1$. Conversely, for any instance of GAP, we can consider the instance of GAP′ having costs $c_{ij} - \bar{c}$ for all $i \in I$ and $j \in J$, where $\bar{c}$ is a sufficiently large constant (e.g., $\bar{c} > 2\sum_{j\in J}(\max_{i\in I} c_{ij} - \min_{i\in I} c_{ij})$). Then, if an optimal solution to the transformed instance of GAP′ satisfies (48.3), the original instance also has an optimal feasible solution whose cost is larger by $n\bar{c}$; otherwise we can conclude that there is no feasible solution to the instance of GAP.

If m is a fixed constant, GAP admits a pseudo polynomial-time exact algorithm, which is based on dynamic programming. GAP with $a_{ij} = 1$ for all $i \in I$ and $j \in J$ becomes a special case of the minimum-cost flow problem and is solvable in polynomial time. (Actually, it is not hard to show that this special case is equivalent to the bipartite maximum weight matching problem.) A more restricted case with $m = n$, $a_{ij} = 1$, and $b_i = 1$ for all $i \in I$ and $j \in J$ is known as the *assignment problem*, and has efficient special algorithms. Efficient algorithms for network flow, matching, and assignment problems are found, e.g., in Refs. [25–27].

GAP can also be formulated as a set partitioning problem. For each agent $i \in I$, suppose that all possible subsets $J' \subseteq J$ (including the empty set) satisfying $\sum_{j\in J'} a_{ij} \leq b_i$ are numbered and let C_i be the set of such indices. A subset $J_i^k \subseteq J$ with $k \in C_i$ represents a set of jobs assigned to agent i. For each J_i^k, define $d_i^k = \sum_{j\in J_i^k} c_{ij}$ and $\delta_{ij}^k = 1$ (respectively, 0) if $j \in J_i^k$ (resp., $j \notin J_i^k$). Then the set partitioning formulation of GAP is given as follows:

$$\text{minimize} \quad \sum_{i\in I}\sum_{k\in C_i} d_i^k y_i^k \tag{48.6}$$

$$\text{subject to} \quad \sum_{i\in I}\sum_{k\in C_i} \delta_{ij}^k y_i^k = 1, \quad \forall j \in J \tag{48.7}$$

$$\sum_{k\in C_i} y_i^k = 1, \quad \forall i \in I \tag{48.8}$$

$$y_i^k \in \{0, 1\}, \quad \forall k \in C_i \text{ and } \forall i \in I \tag{48.9}$$

It is not practical to generate all elements in C_i $(i \in I)$, and hence *column generation* approach is usually used to generate only promising subsets [28,19].

Motivated by practical applications, various generalizations of GAP have been proposed. The *multi resource generalized assignment problem* (MRGAP), in which more than one resource constraint is considered for each agent, is a natural generalization of GAP. For MRGAP, Gavish and Pirkul proposed a branch-and-bound algorithm and two simple Lagrangian heuristics [29]; Rego et al. [30] applied a metaheuristic approach called RAMP (relaxation adaptive memory programming)[31]; and Yagiura et al. [32] devised a very large-scale neighborhood search algorithm. We mention here other generalizations of GAP; e.g., the multi level GAP [10]; the dynamic multi resource GAP [33]; the generalized multi assignment problem [34]; the multi resource GAP with additional constraints [35]; and so forth. The survey by Cattrysse and van Wassenhove [36] discusses some other generalizations of GAP, and summarizes existing results before the early 1990s.

48.3 Greedy Methods

There are several useful tools used to design approximation algorithms. The most common one is perhaps the *greedy method*, which directly constructs approximate solutions by successively determining the values of variables on the basis of some local information. Another important tool is the *local search* (LS), which will be discussed in Section 48.4. In this section, we will describe simple examples of greedy methods for GAP.

The basic framework of greedy methods for GAP is as follows: Start with an empty assignment (i.e., $x_{ij} = 0$ for all $i \in I$ and $j \in J$), and in each iteration, choose a pair of job j and agent i that minimizes an evaluation function $f_{ij}(x)$ among those having sufficient space at agent i (i.e., $\sum_{j' \in J} a_{ij'} x_{ij'} + a_{ij} \leq b_i$), and assign the job j to the agent i (i.e., let $x_{ij} := 1$). This is repeated until a feasible solution is found or it is concluded that the algorithm failed to find a feasible solution. The function $f_{ij}(x)$ measures the desirability of assigning job j to agent i when the current (incomplete) solution is x, where various definitions of $f_{ij}(x)$ are possible. For convenience, let

$$\tilde{b}_i(x) = b_i - \sum_{j' \in J} a_{ij'} x_{ij'}$$

be the currently available amount of resource at agent i and

$$F_j(x) = \left\{ i \in I \mid a_{ij} \leq \tilde{b}_i(x) \right\} \tag{48.10}$$

be the set of agents to which job j can be assigned without violating the resource constraint (48.2). Then the framework of the greedy method is formally described as follows, where $J' \subseteq J$ denotes the set of jobs not assigned yet.

Algorithm GREEDY

Step 1. Let $x_{ij} := 0$ for all $i \in I$ and $j \in J$, and let $J' := J$.

Step 2. If $F_j(x) = \emptyset$ holds for some $j \in J'$, then output “failed” and stop. If $J' = \emptyset$ holds, output the current solution x and stop.

Step 3. Choose a pair (i, j) that minimizes $f_{ij}(x)$ among those satisfying $j \in J'$ and $i \in F_j(x)$. Then let $x_{ij} := 1$, $J' := J' \setminus \{j\}$, and return to Step 2.

Among conceivable functions for $f_{ij}(x)$ are (1) c_{ij}, (2) $(c_{ij} - \bar{c}_j)/a_{ij}$, (3) a_{ij}, and (4) a_{ij}/b_i, where $\bar{c}_j$ is a parameter satisfying $\bar{c}_j \geq \max_{i \in I} c_{ij}$. The evaluation function (2) comes from the profit maximization formulation of GAP, where the counterpart is ρ_{ij}/a_{ij}, the profit per unit resource. It should be noted that this measure ρ_{ij}/a_{ij} is meaningful only if $\rho_{ij} \geq 0$ for all $i \in I$ and $j \in J$.

Below is a more sophisticated way of defining $f_{ij}(x)$, which is based on the concept of “regret” measures. Let f' be a simple evaluation function such as those listed above. For each job $j \in J'$, let i_1 be the agent i in $F_j(x)$ that minimizes $f'_{ij}(x)$. Moreover, if $|F_j(x)| \geq 2$ holds, let i_2 be the agent i that minimizes $f'_{ij}(x)$ among those in $F_j(x) \setminus \{i_1\}$. Then $f_{ij}(x)$ is defined as $f_{i_1 j}(x) = -\infty$ if $|F_j(x)| = 1$, and $f_{i_1 j}(x) = f'_{i_1 j}(x) - f'_{i_2 j}(x)$ otherwise, while $f_{ij}(x) = +\infty$ for $i \neq i_1$. Of course, in the iteration of GREEDY, the x_{ij} with the minimum $f_{ij}(x)$ is chosen and set to $x_{ij} := 1$. This regret measure gives priority to the job having the largest difference in the values of $f'_{ij}(x)$ between the best and the second best agents. While simple original measures are myopic, the regret measures consider one step forward, and hence lead to better performance in many cases. An efficient implementation of such a greedy method is discussed in Ref. [5].

48.4 Local Search

LS is one of the basic tools for designing approximation algorithms. It starts from an initial solution x and repeatedly replaces x with a better solution in its *neighborhood* $N(x)$ until no better solution is found in $N(x)$, where a neighborhood $N(x)$ is a set of solutions obtainable from x with slight perturbations. The resulting solution x is *locally optimal* in the sense that no better solution exists in its neighborhood. LS is sometimes terminated before a locally optimal solution is reached according to some stopping criterion; e.g., when it is computationally expensive to continue the search.

Shift and swap neighborhoods, denoted N_{shift} and N_{swap}, respectively, are commonly used in LS methods for GAP, where

$$\begin{aligned} N_{\text{shift}}(x) &= \{x' \mid x' \text{ is obtainable from } x \text{ by changing the assignment of one job}\}, \\ N_{\text{swap}}(x) &= \{x' \mid x' \text{ is obtainable from } x \text{ by exchanging the assignments of two jobs}\}. \end{aligned}$$

The size of these neighborhoods are $O(mn)$ and $O(n^2)$, respectively. Note that the number of jobs assigned to each agent will not be changed by swap moves; hence it is not appropriate to use the swap neighborhood alone.

The definition of the search space is also an important issue in designing LS. For GAP, it is not effective to search only within the feasible region unless the resource constraints (48.2) are loose, since the problem of determining the existence of a feasible solution is already NP-complete as mentioned in Section 48.2. The following two search spaces are commonly used:

SS1. The set of all possible x that satisfy constraints (48.3) and (48.4) (i.e., the set of all possible assignments $\sigma : J \to I$), implying that the resource constraint (48.2) can be violated during the search.

SS2. The set of all possible x that satisfy constraints (48.2), (48.4), and (48.5) (i.e., the constraint (48.3) is relaxed to (48.5)). This is equivalent to adding a dummy agent "$m + 1$" and considering all possible assignments $\sigma : J \to I \cup \{m + 1\}$ that satisfy the resource constraint (48.2).

We then need to evaluate solutions taking the degree of constraint violation into consideration. An objective function penalized by infeasibility is usually used for this purpose. For example, if the above search space SS1 is used, a natural penalized cost function is as follows:

$$pcost(x) = cost(x) + \sum_{i \in I} \alpha_i \, p_i(x) \tag{48.11}$$

where

$$p_i(x) = \max\left\{0, \ \sum_{j \in J} a_{ij} x_{ij} - b_i\right\}$$

denotes the amount of infeasibility at agent $i \in I$, and α_i (>0) are parameters called *penalty weights*. For the search space SS2, a natural evaluation function is

$$cost(x) + \sum_{j \in J} \beta_j \left(1 - \sum_{i \in I} x_{ij}\right)$$

with appropriate penalty weights β_j (>0). (Equivalently, we can add a dummy agent "$m + 1$" and let $c_{m+1,j} = \beta_j$, $a_{m+1,j} = 1$ for all $j \in J$ and $b_{m+1} = n$, and let the search space to be the feasible region with respect to the modified instance.) The parameters α_i and β_j can be given as fixed constants as in Refs. [37,38], or can be adaptively controlled during the search by using such an algorithm as in [15] (see also Section 48.7.2). A locally optimal solution under the above penalized costs may not always be feasible; however, we can increase the probability of obtaining feasible solutions by (1) keeping the best feasible solution obtained during the search as an incumbent solution and (2) using large values for the penalty weights.

Various algorithms can be realized within the framework of LS by specifying detailed rules in the algorithm. For example, LS with the shift neighborhood in search space SS2 includes the greedy method in Section 48.3, where all jobs are initially assigned to the dummy agent $m + 1$ and shifted to new agents i with $1 \leq i \leq m$ one by one.

As a concrete example of a simple LS algorithm, we introduce below the algorithm MTHG proposed by Martello and Toth [4,5]. This algorithm is often used as a subprocedure for other approximation or exact algorithms. It starts with the greedy method in Section 48.3, using a regret measure as the evaluation function f, and then improves the solution by shift moves, where the feasible shift move with the largest improvement is chosen for each job, and a job will not be shifted again once its assignment is changed; hence the number of shift moves is limited to n. The algorithm is formally described as follows. Recall that $F_j(x)$ defined in (48.10) is the set of agents to which job j can be shifted without violating the resource constraint.

```
Algorithm MTHG
Step 1  Call algorithm GREEDY using a regret measure. If it failed to find a feasible solution, output
        "failed" and stop; otherwise let $x$ be the solution output by GREEDY and $j := 1$.
Step 2  Let $i'$ be the agent to which the job $j$ is currently assigned (i.e., $x_{i'j} = 1$). If $F_j(x)\backslash\{i'\} = \emptyset$
        holds, proceed to Step 3; otherwise let $i''$ be the agent that minimizes $c_{ij}$ among those in $F_j(x)\backslash\{i'\}$.
        If $c_{i''j} < c_{i'j}$ holds, change the assignment of job $j$ from $i'$ to $i''$; i.e., let $x_{i'j} := 0$ and $x_{i''j} := 1$.
Step 3  If $j = n$ holds, output the current solution $x$ and stop; otherwise let $j := j + 1$ and return to
        Step 2.
```

For simplicity, in the above algorithm, shift moves are tested according to the order of their job indices (as described in the same way as in Ref. [5]); however, it is often preferable to shuffle the indices beforehand to avoid undesirable bias. If appropriately implemented, algorithm MTHG runs in $O(nm\log m + n^2)$ time, provided that the simple evaluation f' to be used to compute the regret measure f can be evaluated independently of x as in the case of the four examples discussed in Section 48.3.

48.5 Lagrangian Relaxation

Relaxation problems often provide useful information for designing efficient algorithms to solve the original problem. In this section, we give two types of *Lagrangian relaxation problems* for GAP: One relaxes the resource constraint (48.2), and another relaxes the assignment constraint (48.3). The first type is formally defined as follows:

$$\begin{aligned} L^{\mathrm{rec}}(u) = \min & \sum_{i\in I}\sum_{j\in J}(c_{ij} + u_i a_{ij})x_{ij} - \sum_{i\in I} u_i b_i \\ \text{s.t.} & \sum_{i\in I} x_{ij} = 1, \quad \forall j \in J \\ & x_{ij} \in \{0, 1\}, \quad \forall i \in I \text{ and } \forall j \in J \end{aligned} \tag{48.12}$$

where $u = (u_1, u_2, \ldots, u_m) \in R_+^m$ (R_+ is the set of nonnegative reals) is a Lagrangian multiplier vector given to the resource constraint (48.2). An optimal solution x for this relaxation problem is obtained by the following simple rule: For each job j, let i_j^* be an agent i that minimizes $c_{ij} + u_i a_{ij}$, and let $x_{ij} := 1$ for $i = i_j^*$ and $x_{ij} := 0$ for $i \neq i_j^*$.

For any $u \in R_+^m$, $L^{\mathrm{rec}}(u)$ gives a lower bound on the objective value of problem GAP, and the Lagrangian dual problem is to find a $u \in R_+^m$ that maximizes the lower bound $L^{\mathrm{rec}}(u)$. Any optimal solution $u = u^*$ to the dual of the linear programming (LP) relaxation of GAP,

$$\begin{aligned} LP = \max & \sum_{j\in J} v_j - \sum_{i\in I} b_i u_i \\ \text{s.t.} \quad & v_j - a_{ij}u_i \leq c_{ij}, \quad \forall i \in I \text{ and } \forall j \in J \\ & u_i \geq 0, \quad \forall i \in I \end{aligned} \tag{48.13}$$

is an optimal solution to the Lagrangian dual [39]. However, computing such u^* by solving (48.13) is expensive for large scale instances. Hence the *subgradient method* [39,40] explained below is often used for finding a near-optimal u.

Define the subgradient vector $s(u) \in R^m$ by

$$s_i(u) = \sum_{j \in J} a_{ij} x_{ij}(u) - b_i$$

for all $i \in M$, where $x(u)$ is an optimal solution to $L^{\text{rec}}(u)$. The method generates a sequence of Lagrangian multiplier vectors $u^{(0)}, u^{(1)}, \ldots$, where $u^{(0)}$ is an initial vector, and $u^{(k+1)}$ is updated from $u^{(k)}$ by the following formula:

$$u_i^{(k+1)} := \max\left\{u_i^{(k)} + t_k s_i(u^{(k)}),\ 0\right\}, \quad \forall i \in I$$

where $t_k > 0$ is a scalar called step size. One of the most common rules of determining step sizes is

$$t_k = \lambda_k \frac{UB - L^{\text{rec}}(u^{(k)})}{||s(u^{(k)})||^2}$$

where UB is an upper bound on $cost(x)$ (e.g., obtained by a greedy method or an LS) and $\lambda_k \in (0, 2]$ is a scalar. The sequence of λ_k is often determined by setting $\lambda_0 := 2$ and halving λ_k whenever $\max_{k \geq 0} L^{\text{rec}}(u^{(k)})$ has failed to improve during some number of consecutive iterations.

Another type of Lagrangian relaxation problem is obtained by relaxing the assignment constraint (48.3):

$$\begin{aligned} L^{\text{asgn}}(v) = \min & \sum_{i \in I} \sum_{j \in J} (c_{ij} - v_j) x_{ij} + \sum_{j \in J} v_j \\ \text{s.t.} & \sum_{j \in J} a_{ij} x_{ij} \leq b_i, \quad \forall i \in I \\ & x_{ij} \in \{0, 1\}, \qquad \forall i \in I \text{ and } \forall j \in J \end{aligned} \tag{48.14}$$

where $v = (v_1, v_2, \ldots, v_n) \in R^n$ (R is the set of reals) is a Lagrangian multiplier vector given to the assignment constraint. Problem (48.14) decomposes into m 0-1 knapsack problems, which is also known to be NP-hard [22] but is much easier to solve exactly compared to GAP [5,41]. The Lagrangian dual to problem (48.14) is similarly defined as in the case of (48.12), and the subgradient method is useful also in this case.

For an optimal solution (u^*, v^*) to the linear programming problem (48.13),

$$L^{\text{asgn}}(v^*) \geq L^{\text{rec}}(u^*) = LP$$

holds, and $L^{\text{asgn}}(v^*)$ often provides a tighter lower bound on the optimal value of GAP than $L^{\text{rec}}(u^*) = LP$.

48.6 Lagrangian Heuristics

If a good Lagrangian multiplier vector u (resp., v) is given, an optimal solution x to problem (48.12) (respectively, (48.14)) is often close to be feasible. (If an optimal x to problem (48.14) happens to be feasible, it is optimal for the original GAP, but this property does not hold for problem (48.12).) Hence it is often advantageous to obtain feasible solutions by slightly modifying them. Moreover, the Lagrangian *relative cost* (or *reduced cost*) $c_{ij} + u_i a_{ij}$ and $c_{ij} - v_j$ with respect to the Lagrangian relaxation problems (48.12) and (48.14), respectively, often provide more accurate estimation on the desirability of assigning a job j to an agent i than the original cost c_{ij}. (The relative cost $c_{ij} + u_i a_{ij}$ is suitable for comparing the desirability of agents for a specified job, while the relative cost $c_{ij} - v_j$ is suitable for comparing the desirability of jobs for a specified agent.) A heuristic algorithm based on such information is called a *Lagrangian heuristic algorithm* [39]. For GAP, many such algorithms have been proposed and are confirmed to be quire useful; e.g., Refs. [12,20,21,28,42–44]. Some are based on different relaxation problems from those illustrated in Section 48.5; e.g., Refs. [28,42–44]. Among them, Cattrysse et al. [28] proposed an algorithm based on the set partitioning formulation of GAP. See, e.g. [36], for various types of relaxation problems of GAP. Lagrangian heuristics is also known to be very efficient to solve large-scale instances of other representative combinatorial optimization problems such as location and set covering problems. Below we briefly illustrate basic ideas for such heuristic algorithms.

A standard framework of Lagrangian heuristics is as follows.

> **Lagrangian Heuristic Algorithm**
> For each Lagrangian multiplier vector $u = u^{(k)}$ generated by a subgradient method, if u (or $L^{\text{rec}}(u)$ or $x(u)$) satisfies certain prespecified conditions, construct a feasible solution based on the information from the Lagrangian relaxation problem (48.12). (The algorithm for the Lagrangian relaxation problem [48.14] is similar.)

As the construction of feasible solutions is tried many times in the above iteration, quick methods are preferable. Some such methods will be described below. Moreover, it is not fruitful to construct a feasible solution by using a poor multiplier vector u (e.g., those generated in the early stage of the subgradient method), and hence some conditions are applied to decide whether the construction is executed or not. A simple rule may be to execute only if $(UB - L^{\text{rec}}(u))/L^{\text{rec}}(u)$ is below a prespecified constant.

For the Lagrangian relaxation problem (48.12) with a given u, an optimal solution x to it satisfies the assignment constraint (48.3). It is therefore natural to use such x as the initial solution for an LS algorithm with search space SS1, and improve its feasibility with the shift neighborhood (or the combination of the shift and swap neighborhoods) using sufficiently large penalty weights α_i in $pcost(x)$ of (48.11). Another reasonable method is to call algorithm MTHG in which $f'_{ij}(x) = c_{ij} + u_i a_{ij}$ is used to compute the regret measure.

For the Lagrangian relaxation problem (48.14) with a given v, an optimal solution x to it satisfies the resource constraint (48.2), but may not satisfy the assignment constraint (48.3). A job j may be assigned to two or more agents, or may not be assigned to any agent. Let

$$\begin{aligned} J_0 &= \{j \in J \mid \textstyle\sum_{i\in I} x_{ij} = 0\} \\ J_1 &= \{j \in J \mid \textstyle\sum_{i\in I} x_{ij} = 1\} \\ J_2 &= \{j \in J \mid \textstyle\sum_{i\in I} x_{ij} \geq 2\} \end{aligned}$$

be the set of jobs assigned to no agent, exactly one agent, and more than one agent, respectively. Then it is often the case that $|J_0 \cup J_2|$ is small. Hence it is usually effective to fix the assignment of jobs in J_1 to the currently assigned agents, and try to modify the assignments of jobs in J_0 or J_2 to obtain a feasible solution. A standard method is to remove each job in J_2 from all but one agent, and then assign jobs in J_0 to agents. For choosing an agent for a job j in J_2 among those i with $x_{ij} = 1$, reasonable criteria are (1) c_{ij}, (2) $(c_{ij} - v_j)/a_{ij}$ and so forth, where a smaller value is more preferable. Note that $c_{ij} - v_j < 0$ holds for all $i \in I$ and $j \in J$ satisfying $x_{ij} = 1$ in an optimal solution x to problem (48.14). To assign jobs in J_0 to agents, we can use the greedy method in Section 48.3 or algorithm MTHG to the restricted instances whose free variables are x_{ij} with $j \in J_0$. Once a feasible solution is found, it is often effective to improve it by an LS algorithm. The efficiency of the above method may degrade if $|J_0 \cup J_2|$ is large; hence it is advantageous to try the construction only when $|J_0 \cup J_2|$ is below a prespecified threshold (e.g., [21]).

48.7 Metaheuristics

Metaheuristic algorithms are widely recognized as one of the most practical approaches for hard combinatorial optimization problems. Metaheuristics is a set of comprehensive guidelines or frameworks useful to devise efficient algorithms by combining basic strategies such as greedy methods and LS in a unified manner. Among representative metaheuristic algorithms are genetic algorithms, simulated annealing, tabu search, and so on. See other chapters of this handbook (e.g., 19–21, 23–27) for details of metaheuristic algorithms. While basic strategies such as greedy methods, LS, and Lagrangian heuristics tend to find good solutions quickly, metaheuristic algorithms aim at obtaining better solutions by investing more computation time.

For GAP, many metaheuristic algorithms have been proposed. Among them are a tabu search and a simulated annealing by Osman [6]; a genetic algorithm by Chu and Beasley [7]; variable depth search algorithms by Racer and Amini [8,9]; a tabu search based on ejection chain approach by Laguna et al. [10] (which is proposed for a generalization of GAP); another tabu search by Díaz and Fernández [11]; a MAX-MIN ant system combined with local search and tabu search by Lourenço and Serra [13]; a path-relinking algorithm by Alfandari et al. [14]; variable depth search algorithms [37,38] and ejection chain approaches by Yagiura et al. [15,16]; RAMP (relaxation adaptive memory programming) and primal-dual RAMP algorithms by Rego et al. [30]; and so forth. The algorithm by Rego et al. [30] is one of the latest algorithms, which is based on the RAMP approach recently proposed in [31]. We will illustrate below basic ideas of the ejection chain and path-relinking approaches by Yagiura et al. [15,16], which are among the most efficient metaheuristic algorithms in those listed above.

It utilizes a powerful neighborhood search algorithm [15], called EC probe, based on the idea of *ejection chains* [45]. An ejection chain is an embedded neighborhood construction that compounds simple moves to create more complex and powerful moves. In Ref. [15], EC probe was first incorporated in the framework of tabu search, and promising results were obtained. We call this algorithm tabu search with ejection chains (TSEC). In Ref. [16], EC probe was then incorporated with the *path relinking* approach, which provides an "evolutionary" mechanism for generating new solutions by combining two or more reference solutions. We call the resulting algorithm *PREC* (path relinking with ejection chains).

As mentioned in Section 48.4, it is difficult to conduct the search only within the feasible region. Algorithms TSEC and PREC are both based on the search space SS1, and evaluate solutions by the penalized cost (48.11). As the search is sensitive to penalty weights α_i, the algorithms employ a sophisticated adaptive control mechanism of the penalty weights, thereby introducing strategic oscillation mechanism in the algorithms. This adaptive control method improves the performance and robustness of the algorithms as well as their usability.

In the following subsections, we briefly explain the idea of EC probe and adaptive control of penalty weights, and then summarize the frameworks of TSEC and PREC. For convenience, in the rest of this section, we use an assignment σ instead of x to represent a solution, and use notations such as $pcost(\sigma)$ instead of $pcost(x)$, as their meanings are clear.

48.7.1 EC Probe

An EC move in algorithms TSEC and PREC is a sequence of shift moves such that a solution σ' is obtained from the current solution σ by shifting l ($l = 2, 3, \ldots, n$) jobs $j_1, j_2, \ldots, j_l$ simultaneously in such a way that satisfies $\sigma'(j_r) = \sigma(j_{r-1})$ for $r = 2, 3, \ldots, l$, where $\sigma'(j_1)$ is arbitrary. In other words, for $r = 2, 3, \ldots, l$, job j_r is shifted from agent $\sigma(j_r)$ to agent $\sigma(j_{r-1})$ after ejecting job j_{r-1}. An EC move is called *cyclic* if $\sigma'(j_1) = \sigma(j_l)$ holds, i.e., the first job j_1 is inserted into the agent from which the last job j_l is ejected. The *length* of an EC move is the number of shift moves l in the sequence. The EC neighborhood is the set of solutions obtainable by such EC moves. Both shift and swap neighborhoods are subsets of the EC neighborhood, since a shift move is an EC move of length 1, and a swap move is a cyclic EC move of length 2. However, the size of the EC neighborhood can become exponential unless intelligently controlled. For this purpose, three subsets of the neighborhood called *shift*, *double shift*, and *long chain* are considered, where a double-shift move is an EC move of length 2, and a long-chain move is an EC of any length. For the double-shift neighborhood, the number of candidates for j_2 is restricted to $\max\{m, \log n\}$ when j_1 is fixed, and for the long-chain neighborhood, only one candidate for j_l is considered when $j_1, j_2, \ldots, j_{l-1}$ are fixed. Such candidates are chosen by using heuristic rules based on the observation that given a good multiplier vector v, the Lagrangian relative cost $c_{ij} - v_j$ for Lagrangian relaxation (48.14) tends to represent desirability of assigning job j to agent i as discussed in Section 48.6. More details of such rules are found in Ref. [15].

As a result, the sizes of shift, double-shift, and long-chain neighborhoods become $O(mn)$, $O(n \max \{m, \log n\})$, and $O(n^2)$, respectively. Yagiura et al. [15] also showed that the expected size of the long-chain neighborhood was $O(n^{(3/2)+\varepsilon})$ for an arbitrarily small positive ε under a simplified random model. These three neighborhoods are used alternately to form an improving phase of LS, which is called *EC probe*.

48.7.2 Adaptive Control of the Penalty Weights

In this section, we briefly explain the basic idea of the adaptive control of the penalty weights given in Refs. [15,16]. The initial values of α_i are decided by solving a quadratic programming problem (QP), whose main aim is to balance the estimated change in the cost and penalty after shift moves. The definition of the QP in Ref. [15] is slightly complicated and is omitted here. Then, whenever an EC probe stops at a locally optimal solution σ_{lopt}, α_i values are updated by the following rule. If no feasible solution is found during the previous EC probe after the last update of α_i, the penalty weights are increased by

$$\alpha_i := \alpha_i \left(1 + \delta_{\text{inc}} \cdot \frac{p_i(\sigma_{\text{lopt}})/b_i}{\max_{i' \in I}(p_{i'}(\sigma_{\text{lopt}})/b_{i'})}\right)$$

for all $i \in I$, where δ_{inc} is a parameter. Otherwise (i.e., if at least one feasible solution is found during the previous EC probe after the last update of α_i), the penalty weights are decreased by

$$\alpha_i := \alpha_i(1 - \delta_{\text{dec}})$$

for all $i \in I$ that satisfy $p_i(\sigma_{\text{lopt}}) = 0$, where δ_{dec} is a parameter. In the computational experiments in Section 48.9, δ_{inc} and δ_{dec} are set to 0.01 and 0.1, respectively, as in Refs. [15,16].

48.7.3 Algorithms TSEC and PREC

Algorithm TSEC applies an EC probe to a solution obtained by perturbing an available good solution σ_{seed}. Solution σ_{seed} is initially generated randomly, and is replaced with the locally optimal solution σ_{lopt} obtained by the previous EC probe whenever $pcost(\sigma_{\text{lopt}}) \leq pcost(\sigma_{\text{seed}})$ holds. The perturbed solution to which the next EC probe is applied is determined by first generating solutions by applying shift moves to σ_{seed} and then choosing the one that minimizes *pcost* among those not tested yet from the current σ_{seed}. Note that the perturbed solution σ generated by applying a shift move to σ_{seed} is improved first by the cyclic double-shift neighborhood in EC probe. This strategy was motivated by the success of SSS probe [37], and was confirmed to be effective to avoid short cycling. The idea is based on the fact that a cyclic move does not change the number of jobs assigned to each agent, while a shift move changes those of relevant agents. See Ref. [15] for more discussion.

Algorithm PREC applies EC probes to solutions generated by path relinking, which is a methodology to generate solutions from two or more solutions. It keeps a reference set R ($|R| = \pi$ is a parameter) of good solutions. Initially, R is prepared by applying EC probes to randomly generated solutions. Then it is updated by reflecting the outcomes of LS. The incumbent solution (i.e., the best feasible solution) is always stored as a member of R. Other solutions in R are maintained as follows. Whenever an EC probe stops, the locally optimal solution σ_{lopt} is exchanged with the worst (with respect to *pcost*) solution σ_{worst} in R (excluding the incumbent solution), provided that σ_{lopt} is not worse than σ_{worst} and is different from all solutions in R. Path relinking is applied to two solutions σ_{A} (initiating solution) and σ_{B} (guiding solution) randomly chosen from R, where a random shift is applied to σ_{B} with probability 1/2 (no shift with the remaining probability 1/2) before applying the path relinking (for the purpose of keeping the diversity of the search), and the resulting solution is redefined to be σ_{B}. Let the distance between two solutions σ and σ' be $dist(\sigma, \sigma') = \left|\{j \in J \mid \sigma(j) \neq \sigma'(j)\}\right|$, i.e., the number of jobs assigned to different agents, and let $d = dist(\sigma_{\text{A}}, \sigma_{\text{B}})$ be the distance between solutions σ_{A} and σ_{B}. The algorithm generates a sequence $\sigma_0, \sigma_1, \ldots, \sigma_d$ of solutions from two solutions σ_{A} and σ_{B} as follows. Starting from $\sigma_0 := \sigma_{\text{A}}$, for $k = 1, 2, \ldots, d$, let σ_k be the solution in $N_{\text{shift}}(\sigma_{k-1})$ with the best *pcost* among those whose distances to σ_{B} are smaller than that from σ_{k-1}. Let S be the best γ (a parameter) solutions with respect to *pcost* from

$\{\sigma_2, \sigma_3, \ldots, \sigma_{d-1}\}$.[1] EC probes are then applied to all solutions in S, which completes this phase of path relinking. For each starting solution $\sigma \in S$, it is improved first by the cyclic double-shift neighborhood in EC probe to avoid short cycling, as in algorithm TSEC.

Yagiura et al. [16] also tested other rules, e.g., perturbing guiding solutions σ_{B} or not, using more than one guiding solution for generating a path, restricting the distance between solutions in the reference set, rules to choose initiating and guiding solutions, and rules to define the set S, and observed that the algorithm was not sensitive to such changes. In the computational experiment in Section 48.9, the parameters in PREC were set to $\pi = 20$ and $\gamma = 10$ as in Ref. [16].

48.8 Branch-and-Bound Algorithms

Branch-and-bound is one of the major enumerative methods to solve combinatorial optimization problems. It is based on the idea that the original problem can be equivalently solved as a result of solving all partial problems decomposed from the original problem. The decomposition can then be applied to the generated partial problems. A problem instance of GAP can be decomposed into two partial problems by fixing some variable x_{ij} to 0 and 1, where such a variable is called a *branching variable*, and the corresponding operation is called a *branching operation*. We denote the original problem by P_0 and the kth partial problem generated during computation by P_k. Starting from P_0, if we apply branching operations repeatedly until all partial problems become trivial, we can enumerate all possible solutions. During this process, we usually examine only a small portion of all partial problems, based on the following idea. If an optimal solution to a partial problem P_k is found or it is concluded for some reason that an optimal solution to the original problem can be found even without solving P_k, then it is not necessary to consider P_k further. The operation of removing such a P_k from the list of partial problems to be solved is called a *bounding operation*, and we say that this operation *terminates* P_k. For example, denoting the incumbent value by *UB*, and a lower bound on the objective value for P_k by $LB(P_k)$, we can terminate P_k if $LB(P_k) \geq UB$ holds. This is called a *lower bound test*. The process of branching operations can be expressed by a search tree, whose root corresponds to P_0, and the children of a node correspond to the partial problems generated by a branching operation applied to the node. A partial problem is called *active* if it has not been terminated or decomposed into partial problems. A branch-and-bound algorithm obtains an exact optimal solution if no active partial problem remains when it stops. The rule to choose an active partial problem to test is called the *search strategy*, which affects the efficiency of the search.

A common way for obtaining a lower bound *LB* is to solve relaxation problems such as the Lagrangian relaxations (48.12) and (48.14) in Section 48.5, and the LP relaxation of GAP in which the 0-1 constraint $x_{ij} \in \{0, 1\}$ of (48.4) is relaxed to $x_{ij} \geq 0$ for $\forall i \in I$ and $\forall j \in J$. (Though $0 \leq x_{ij} \leq 1$ seems more natural, $x_{ij} \leq 1$ is redundant because of the assignment constraint (48.3).) As mentioned in Section 48.5, the lower bound $L^{\mathrm{asgn}}(v)$ from problem (48.14) tends to be better than the bound $L^{\mathrm{rec}}(u)$ from (48.12) or the lower bound from the LP relaxation. Computing better lower bounds will result in reducing the number of partial problems generated during the search, and hence the bound $L^{\mathrm{asgn}}(v)$ is often used in branch-and-bound algorithms. Another common approach to obtain better lower bounds is the use of *valid inequalities*, where a linear inequality constraint is called valid if all feasible solutions to the original GAP satisfy it (but solutions to the LP relaxation problem may not satisfy it). The optimal values of LP relaxations with such additional constraints will usually give better lower bounds. Branch-and-bound algorithms using such valid inequalities are called *branch-and-cut* methods. We give below a simple example of a valid inequality. Let $\bar{x}$ be an optimal solution to the LP relaxation of GAP, and let $i \in I$ be an agent having at least one $\bar{x}_{ij}$ fractional (i.e., $0 < \bar{x}_{ij} < 1$). Sort the jobs j with $\bar{x}_{ij} > 0$ in the nonincreasing order of a_{ij}, and let $j_1, j_2, \ldots$ be the resulting order (i.e., $a_{ij_1} \geq a_{ij_2} \geq \cdots$). Let k be the index that satisfies $\sum_{l=1}^{k-1} a_{ij_l} \leq b_i < \sum_{l=1}^{k} a_{ij_l}$, and let $J_i = \{j_1, j_2, \ldots, j_k\}$. Then $\sum_{j \in J_i} x_{ij} \leq |J_i| - 1$ is a valid

[1] A part of their neighbors are also considered as candidates for S.

inequality. Therefore, if the solution $\bar{x}$ violates it (i.e., $\sum_{j \in J_i} \bar{x}_{ij} > |J_i| - 1$), it will be useful to improve the lower bound.

The objective value of any feasible assignment is an upper bound on the optimal value. We can use approximation algorithms to obtain good bounds *UB*; e.g., greedy methods, LS, Lagrangian heuristics, or metaheuristic algorithms. Having a good *UB* at an early stage is useful for reducing computation time, because many partial problems may be terminated by lower bound tests. It is often effective to apply approximation algorithms to partial problems with small lower bounds. Lagrangian heuristic algorithms are often used for this purpose when Lagrangian relaxation is used to compute lower bounds.

While branch-and-bound algorithms are usually designed to obtain exact optimal solutions, they can also be used as approximation algorithms if the computation is cut off before normal termination. Approximation algorithms of this type are sometimes effective, especially when the size of the problem instance is not very large as will be observed in Section 48.9. One of the merits of such an approximation branch-and-bound is that the minimum of $LB(P_k)$ among all active P_ks gives a lower bound that tends to be better than the lower bound to the original problem $LB(P_0)$. Search strategy to find good upper bound earlier may be different from the one that makes the exact branch-and-bound algorithm efficient. Hence it will be worth trying to design a good search strategy for the purpose of obtaining a good approximation branch-and-bound algorithm.

Among early papers on branch-and-bound algorithms for GAP are Refs. [4,17,18]. Cattrysse and van Wassenhove [36] surveyed various relaxation problems used to compute lower bounds for branch-and-bound methods, such as Lagrangian relaxation, surrogate relaxation, and so on. Recently, Savelsbergh [19] specialized a general-purpose solver, MINTO, to solve GAP based on a branch-and-price approach to the set partitioning formulation of GAP. Nauss [20] incorporated various ideas to improve lower bounds and fixing variables, and obtained impressive results. Haddadi and Ouzia [21] also gave good results with a relatively simple branch-and-bound algorithm. See a recent survey by Nauss [46] for more information on branch-and-bound algorithms for GAP.

48.9 Computational Results

In this section, we show computational results of some metaheuristic algorithms and branch-and-bound algorithms. There are five types of benchmark instances called types A, B, C, D, and E [4,7,10]. Out of threse, we use three types C, D, and E, since types A and B are relatively easy. Instances of these types are generated as follows:

Type C: a_{ij} are random integers from [5, 25], c_{ij} are random integers from [10, 50], and $b_i = 0.8 \sum_{j \in J} a_{ij}/m$.
Type D: a_{ij} are random integers from [1, 100], $c_{ij} = 111 - a_{ij} + e_{ij}$, where e_{ij} are random integers from [−10, 10], and $b_i = 0.8 \sum_{j \in J} a_{ij}/m$.
Type E: $a_{ij} = 1 - 10 \ln e_{ij}$, where e_{ij} are random numbers from (0,1], $c_{ij} = 1000/a_{ij} - 10\hat{e}_{ij}$, where $\hat{e}_{ij}$ are random numbers from [0, 1], and $b_i = 0.8 \sum_{j \in J} a_{ij}/m$.

Types D and E are somewhat harder than other types, since c_{ij} and a_{ij} are inversely correlated. Note that types D and E instances should be solved as minimization problems, since otherwise they are trivial. Among the three sets of problem instances SMALL, MEDIUM, and LARGE used in Ref. [15], we use set MEDIUM in this section, which are total of 18 instances of types C, D, and E with n up to 200, each type consisting of six instances. Among them, types C and D instances were taken from OR-Library,[2] and type E instances were generated by us, which are available at our site.[3]

[2] URL of OR-Library: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html

[3] URL of our site: http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/gap/

TABLE 48.1 The Best Costs Obtained by the Tested Algorithms

Type	n	m	LB	PREC	TSEC	BVDS-l	BVDS-j	VDS	RA	LKGG	NI	RLS[a]	CB[b]	DF[c]	CPLEX
C	100	5	1930	1931[a]	1931[a]	1931[a]	1931[a]	1931[a]	1938	1931[a]	1931[a]	1942	1931[a]	1931[a]	1931[a]
C	100	10	1400	1402[a]	1402[a]	1402[a]	1403	1402[a]	1405	1403	1403	1407	1403	1402[a]	1402[a]
C	100	20	1242	1243[a]	1243[a]	1244	1244	1246	1250	1245	1245	1247	1244	1243[a]	1243[a]
C	200	5	3455	3456[a]	3456[a]	3456[a]	3457	3457	3469	3457	3465	3467	3458	3457	3456[a]
C	200	10	2804	2807	2806[a]	2809	2808	2809	2835	2812	2817	2818	2814	2807	2806[a]
C	200	20	2391	2391[a]	2392	2401	2400	2405	2419	2396	2407	2405	2397	2391[a]	2391[a]
D	100	5	6350	6353[a]	6357	6358	6362	6365	—	6386	6415	6476	6373	6357	6358
D	100	10	6342	6356	6358	6367	6370	6380	6532	6406	6487	6469	6379	6355[a]	6381
D	100	20	6177	6211[a]	6221	6275	6245	6284	6428	6297	6368	6358	6269	6220	6280
D	200	5	12741	12744[a]	12746	12755	12755	12778	—	12788	12973	12923	12796	12747	12750
D	200	10	12426	12438[a]	12446	12480	12473	12496[d]	12799	12537	12889	12746	12601	12457	12457
D	200	20	12230	12269[a]	12284	12440	12318	12335[d]	12665	12436	12793	12617	12452	12351	12393
E	100	5	12673	12681[a]	12682	12681[a]	12682	12685	12917	12687[e]	12686[f]	12836	N. A.	12681[a]	12681[a]
E	100	10	11568	11577	11577[a]	11585	11599	11585	12047	11641[e]	11590[f]	11780	N. A.	11581	11593
E	100	20	8431	8444	8443[a]	8499	8484	8490	9004	8522[e]	8509[f]	8717	N. A.	8460	8565
E	200	5	24927	24930[a]	24930[a]	24942	24933	24948	25649	25147[e]	24958[f]	25317	N. A.	24931	24930[a]
E	200	10	23302	23310	23307[a]	23346	23348	23340	24717	23567[e]	23396[f]	23620	N. A.	23318	23321
E	200	20	22377	22379[a]	22391	22475	22437	22452[d]	24117	22659[e]	22551[f]	22779	N. A.	22422	22457

Note: The time limits are 150 and 300 s for $n = 100$ and 200, respectively; one execution per instance except RLS, CB and DF.

[a] Computation time is reported in Ref. [15].

[b] Results in Ref. [7].

[c] Results in Ref. [11].

[d] Results after 1,000 seconds on Sun Ultra 2 Model 2300.

[e] Results after 20,000 seconds on Sun Ultra 2 Model 2300.

[f] Results after 5,000 seconds on Sun Ultra 2 Model 2300.

We first compare in Table 48.1 various metaheuristic algorithms: (1) variable depth search by Yagiura et al. [38] (denoted VDS), (2) two algorithms of branching variable depth search by Yagiura et al. [37] (denoted BVDS-l and BVDS-j), (3) tabu search based on ECs by Yagiura et al. [15] (denoted TSEC), (4) path relinking approach based on ECs by Yagiura et al. [16] (denoted PREC), (5) variable depth search by Racer and Amini [9] (denoted RA), (6) tabu search by Laguna et al. [10] (denoted LKGG), (7) tabu search for the general purpose constraint satisfaction problem by Nonobe and Ibaraki [47] (denoted NI), (8) a MAX-MIN ant system combined with LS and tabu search by Lourenço and Serra [13] (denoted RLS), (9) the genetic algorithm by Chu and Beasley [7] (denoted CB), and (10) the tabu search by Díaz and Fernández [11] (denoted DF). The results of a commercial solver CPLEX 6.5[4] are also shown in the column CPLEX. All the data were taken from Table 2 of Ref. [16]. Unless otherwise stated, algorithms were run on a workstation Sun Ultra 2 Model 2300 (two UltraSPARC II 300MHz processors with 1 GB memory), where the computation was executed on a single processor. SPECint95 of this workstation is 12.3 according to the SPEC site[5] (Standard Performance Evaluation Corporation). The table shows the best costs obtained by the algorithms within 150 s for $n = 100$, and 300 s for $n = 200$, respectively, unless otherwise stated below the table. The computation time of RLS, CB, and DF is longer than this time limit as discussed in Refs. [15,16]. Table 48.1 also shows the lower bounds (denoted LB) obtained by solving the Lagrangian relaxation (48.14) of GAP. Each "$*$" mark indicates that the best cost is attained, and "—" means that no feasible solution was found. The results of algorithm CB for type E instances are not available (i.e., not reported in their paper), and are denoted "N. A." in the table. Note that all algorithms except CB and DF were run only once for each instance in the computational results in this section in order to make the

[4] CPLEX 8.1.0 was also tested, but the results of CPLEX 6.5 were slightly better on average.

[5] URL of SPEC site: http://www.spec.org/

TABLE 48.2 Results of Branch-and-Bound Algorithms by Nauss [20] (NaussBB) and Haddadi and Ouzia [21] (HOBB)

				NaussBB				HOBB			
Type	n	m	Best Known	Best	Time to Best	Total Time	Opt?	Best	Time to Best	Total Time	Opt?
C	100	5	1931	1931	0.1	3.7	yes	1931	0.8	7.5	yes
C	100	10	1402	1402	7.3	15.1	yes	1402	1.2	20.6	yes
C	100	20	1243	1243	90.2	115.2	yes	1243	22.2	26.8	yes
C	200	5	3456	3456	30.4	40.6	yes	3456	27.7	34.9	yes
C	200	10	2806	2806	312.6	490.4	yes	2806	177.0	193.3	yes
C	200	20	2391	2391	968.7	1028.3	yes	2391	94.4	96.0	yes
D	100	5	6353	6353	349.9	362.2	yes	6353	65.5	471.0	yes
D	100	10	6348	6349	2831.5	T. O.	no	6349	371.6	T. O.	no
D	100	20	6190	6200	2829.4	T. O.	no	6196	17.8	T. O.	no
D	200	5	12742	12745	2937.0	T. O.	no	12742	535.7	1481.5	yes
D	200	10	12432	12447	1896.8	T. O.	no	12436	2068.1	T. O.	no
D	200	20	12241	12263	2375.4	T. O.	no	12250	55.6	T. O.	no

Note: NaussBB is with a time limit of 3000 s on a Pentium II 300 MHz, and HOBB with a time limit 2400 s on a Pentium MMX 200 MHz.

comparison fair. From the table, we can observe that PREC and TSEC are highly effective, especially for the type D and E instances. They obtained the best solutions for most of the tested instances.

We next show the results of recent branch-and-bound algorithms by Nauss [20] (denoted NaussBB) and Haddadi and Ouzia [21] (denoted HOBB) in Table 48.2. The results of NaussBB were taken from Tables 1 and 5 of Ref. [20], and those of HOBB were taken from Table 4 of Ref. [21]. (Table 4 of Ref. [21] also includes the results by NaussBB, but they seem to be taken from Table 4 of Ref. [15], which are slightly different from those in Ref. [20] because the experiments were conducted under different settings as mentioned in Ref. [15].) NaussBB was coded in FORTRAN 77 and run on a Dell XPS D300 (Pentium II 300 MHz with a 64-MB memory), while HOBB was coded in Turbo-Pascal 7.0 and run on an IBM compatible PC (Pentium MMX 200 MHz). SPECint95 of these computers are 11.9 and 6.4, respectively, according to the SPEC site; hence the speed of the Dell XPS D300 is approximately the same as the Sun Ultra 2, while the IBM-compatible PC is slower approximately by a factor of 2. The time limit of NaussBB was 3000 s, while that of HOBB was 2400 s. The column "Best" shows the best cost obtained by the algorithm within the time limit. The column "Time to Best" shows the computation time used to obtain the solution reported in column "Best." The column "Total Time" shows the whole computation time needed for the algorithm until it stops after confirming optimality, where "T. O." signifies that the algorithm stopped when the prespecified time limit was reached before confirming optimality. If the algorithm stopped after confirming optimality, the column "Opt?" is "yes"; otherwise the column is "no." The column "Best Known" shows the best known values reported in Table 6 of Ref. [16].

From the table, we can observe the following: (1) type C instances with up to $n = 200$ can be solved exactly within a reasonable amount of computation time; and (2) type D instances are much harder to solve exactly, but the best results obtained by the branch-and-bound methods are of high quality. Moreover, the computation time of HOBB to obtain such solutions are reasonably small in most cases compared with the time limit of Table 48.1, which implies the effectiveness of the algorithm when it is used as heuristics.

In these results, we observed that algorithms PREC, TSEC, and HOBB are highly efficient. However, it is not safe to draw decisive conclusions from limited computational results. (We reported computational results for these instances mainly because the results of many algorithms are available for these sizes, while very limited results have been reported for larger instances to the best of our knowledge.) For more extensive experimental results, readers are referred to Ref. [16], in which results of some metaheuristic algorithms on benchmark instances with up to $n = 1600$ and $m = 80$ with various time limits are reported, and those instances are available from the authors' web site.

48.10 Performance Guarantee

In this section, we briefly summarize theoretical results on the performance guarantees of approximation algorithms for GAP. As mentioned in Section 48.2, determining the existence of a feasible solution is already NP-complete; hence no polynomial-time approximation algorithm is possible unless some assumptions are made or a different formulation is considered.

We first consider the case where the resource constraint (48.2) can be violated. Without loss of generality, we assume in this section that the available amount of resource at agents satisfy $b_1 = b_2 = \cdots = b_m = \bar{b}$ by scaling the amount of resource. Lin and Vitter [48] gave a polynomial-time algorithm that—given cost $\bar{c}$, amount of resource $\bar{b}$, and any constant $\varepsilon > 0$—finds a solution of cost at most $(1 + \varepsilon)\bar{c}$ where the required amount of resource at each agent is at most $(2 + 1/\varepsilon)\bar{b}$, if there exists a feasible solution of cost at most $\bar{c}$. Shmoys and Tardos [49] showed that there is a polynomial-time algorithm that, given a value $\bar{c}$, either proves that no feasible solution of cost $\bar{c}$ exists, or else finds a solution of cost at most $\bar{c}$ where the resource requirement at each agent is at most $2\bar{b}$. On the negative side, Lenstra et al. [50] considered the problem of minimizing $\bar{b}$ (i.e., the assignment cost is not considered and $\bar{b}$ becomes the objective value instead of being a given constant), and showed that for any $\alpha < 3/2$, no polynomial-time α-approximation algorithm exists unless P = NP, where an α-approximation algorithm guarantees to produce a solution with objective function value at most (respectively, at least) α times the optimum for a minimization (respectively, maximization) problem. Shmoys and Tardos also considered the problem of minimizing the weighted sum of the assignment cost $\sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij}$ and the required amount of resource $\bar{b}$ at each agent, and gave a polynomial-time 2-approximation algorithm.

We next consider another variant of GAP, in which the objective is to maximize the total profit $\sum_{i\in I}\sum_{j\in J} \rho_{ij}x_{ij}$ and the assignment constraint (48.3) is relaxed to (48.5), i.e., $\sum_{i\in I} x_{ij} \le 1$ $(\forall j \in J)$, and call it the max-profit GAP (or MPGAP in short). As discussed in Section 48.2, this problem is equivalent to the original GAP when the objective is to find an exact optimal solution; however, MPGAP is suitable for considering approximation algorithms with performance guarantees because it always has a feasible solution while the original GAP may not. Chekuri and Khanna [23] and Nutov et al. [24] showed that highly restricted cases of MPGAP are APX-hard.

On the positive side, Chekuri and Khanna showed that a polynomial-time 1/2-approximation algorithm exists, which is based on the above-mentioned result by Shmoys and Tardos, and is explained as follows. The theorem by Shmoys and Tardos can be restated as follows. (Though they did not state some of the following facts explicitly, they are immediate from the proof in [49] as pointed out in [23]).

Theorem 48.1 (Shmoys and Tardos [49])

Given an instance of GAP that has a feasible solution of cost $\bar{c}$, there is a polynomial-time algorithm that produces a solution x that satisfies the following three conditions.

1. *The cost of the solution x is at most $\bar{c}$.*
2. *Each job j assigned to an agent i satisfies $a_{ij} \le b_i$.*
3. *If $\sum_{j\in J} a_{ij}x_{ij} > b_i$ holds for an agent $i \in I$, there exists a job j' that satisfies $\sum_{j\in J\setminus\{j'\}} a_{ij}x_{ij} \le b_i$.*

Then the algorithm of Chekuri and Khanna works as follows. Given an instance of MPGAP whose optimal profit is ρ^*, obtain an equivalent instance of GAP as explained in Section 48.2, and call the algorithm of Shmoys and Tardos. Then we have a solution x that has profit at least ρ^* and satisfies the above conditions 2 and 3. For each i with $\sum_{j\in J} a_{ij}x_{ij} > b_i$, let j' be a job that satisfies $\sum_{j\in J\setminus\{j'\}} a_{ij}x_{ij} \le b_i$, and let $x_{ij'} := 0$ if $\rho_{ij'} < \sum_{j\in J} \rho_{ij}x_{ij}/2$, or let $x_{ij} := 0$ for all $j \ne j'$ otherwise. Then the obtained solution is feasible, and has profit at least $\rho^*/2$.

Nutov et al. [24] considered the case with $\rho_{ij} = \rho_j$ $(\forall j \in J)$, and showed that a polynomial-time $(1 - 1/e)$-approximation algorithm exists, where e is the base of the natural logarithm function. When an instance of MPGAP satisfies $\rho_{ij} = \rho_j$ and $a_{ij} = a_j$ $(\forall j \in J)$, the problem is called the *multiple knapsack* *problem* (MKP). Chekuri and Khanna proposed a PTAS for MKP, and showed that there is no FPTAS for MKP even with $m = 2$ unless P = NP.

48.11 Conclusion

In this chapter, we reviewed various algorithms for GAP, such as greedy methods, LS, Lagrangian heuristics, metaheuristics, and branch-and-bound approaches. As examples of efficient methods, basic components of recent metaheuristic algorithms by Yagiura et al. [15,16] were explained. We then gave some computational comparisons of representative metaheuristic algorithms as well as some branch-and-bound methods. We also surveyed performance guarantees of some polynomial-time approximation algorithms. The survey in this chapter is by no means comprehensive, but we hope this article gives useful information for people who are interested in devising efficient algorithms to solve this basic problem, which is of practical as well as theoretical importance.

References

[1] Mazzola, J. B., Neebe, A. W., and Dunn, C. V. R., Production planning of a flexible manufacturing system in a material requirements planning environment, *Int. J. Flexible Manuf. Syst.*, 1/2, 115, 1989.

[2] Ross, G. T. and Soland, R. M., Modeling facility location problems as generalized assignment problems, *Manage. Sci.*, 24, 345, 1977.

[3] Fisher, M. L. and Jaikumar, R., A generalized assignment heuristic for vehicle routing, *Networks*, 11, 109, 1981.

[4] Martello, S. and Toth, P., An algorithm for the generalized assignment problem, in *Proc. IFORS Int. Conf. Oper. Res.*, Brans, J. P., ed., North-Holland, Amsterdam, 1981, p. 589.

[5] Martello, S. and Toth, P., *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, Chichester, 1990.

[6] Osman, I. H., Heuristics for the generalized assignment problem: simulated annealing and tabu search approaches, *OR Spektrum*, 17, 211, 1995.

[7] Chu, P. C. and Beasley, J. E., A genetic algorithm for the generalized assignment problem, *Comput. Oper. Res.*, 24, 17, 1997.

[8] Amini, M. M. and Racer, M., A hybrid heuristic for the generalized assignment problem, *Eur. J. Oper. Res.*, 87, 343, 1995.

[9] Racer, M. and Amini, M. M., A robust heuristic for the generalized assignment problem, *Ann. Oper. Res.*, 50, 487, 1994.

[10] Laguna, M., Kelly, J. P., González-Velarde, J. L., and Glover, F., Tabu search for the multilevel generalized assignment problem, *Eur. J. Oper. Res.*, 82, 176, 1995.

[11] Díaz, J. A. and Fernández, E., A tabu search heuristic for the generalized assignment problem, *Eur. J. Oper. Res.*, 132, 22, 2001.

[12] Haddadi, S. and Ouzia, H., An effective Lagrangian heuristic for the generalized assignment problem, *INFOR*, 39, 351, 2001.

[13] Lourenço H. R. and Serra, D., Adaptive search heuristics for the generalized assignment problem, *Mathware Soft Comput.*, 9, 209, 2002.

[14] Alfandari, L., Plateau, A., and Tolla, P., A path relinking algorithm for the generalized assignment problem, in *Metaheuristics: Computer Decision-Making*, Resende, M. G.C., and de Sousa, J. P., Eds., Kluwer Academic Publishers, Boston, MA, 2003, p. 1.

[15] Yagiura, M., Ibaraki, T., and Glover, F., An ejection chain approach for the generalized assignment problem, *INFORMS J. Comput.*, 16, 133, 2004.

[16] Yagiura, M., Ibaraki, T., and Glover, F., A path relinking approach with ejection chains for the generalized assignment problem, *Eur. J. Oper. Res.*, 169, 548, 2006.

[17] Ross, G. T. and Soland, R. M., A branch and bound algorithm for the generalized assignment problem, *Math. Prog.*, 8, 91, 1975.

[18] Fisher, M. L., Jaikumar, R., and van Wassenhove, L. N., A multiplier adjustment method for the generalized assignment problem, *Manage. Sci.*, 32, 1095, 1986.

[19] Savelsbergh, M., A branch-and-price algorithm for the generalized assignment problem, *Oper. Res.*, 45, 831, 1997.

[20] Nauss, R. M., Solving the generalized assignment problem: An optimizing and heuristic approach, *INFORMS J. on Comput.*, 15, 249, 2003.

[21] Haddadi, S. and Ouzia, H., Effective algorithm and heuristic for the generalized assignment problem, *Eur. J. Oper. Res.*, 153, 184, 2004.

[22] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[23] Chekuri, C. and Khanna, S., A polynomial time approximation scheme for the multiple knapsack problem, *SIAM J. Comput.*, 35, 713, 2006.

[24] Nutov, Z., Beniaminy, I., and Yuster, R., A $(1 - 1/e)$-approximation algorithm for the generalized assignment problem, *Oper. Res. Lett.*, 34, 283, 2006.

[25] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Upper Saddle River, NJ, 1993.

[26] Korte, B. and Vygen, J., *Combinatorial Optimization: Theory and Algorithms*, 3rd ed., Springer, Berlin, 2006.

[27] Schrijiver, A., *Combinatorial Optimization: Polyhedra and Efficiency*, Springer, Berlin, 2003.

[28] Cattrysse, D. G., Salomon, M., and van Wassenhove, L. N., A set partitioning heuristic for the generalized assignment problem, *Eur. J. Oper. Res.*, 72, 167, 1994.

[29] Gavish, B. and Pirkul, H., Algorithms for the multi-resource generalized assignment problem, *Manage. Sci.*, 37, 695, 1991.

[30] Rego, C., Sagbansua, L., Alidaee, B., and Glover, F., RAMP and Primal-dual RAMP Algorithms for the Multi-Resource Generalized Assignment Problem, Research Report, Hearin Center for Enterprise Science, School of Business Administration, University of Mississippi, 2005.

[31] Rego, C., RAMP: A new metaheuristic framework for combinatorial optimization, in *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, Rego, C. and Alidaee, S., Eds., Kluwer Academic Publishers, Dordrecht, 2005, p. 441.

[32] Yagiura, M., Iwasaki, S., Ibaraki, T., and Glover, F., A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem, *Discrete Optimization*, 1, 87, 2004.

[33] Shtub, A. and Kogan, K., Capacity planning by the dynamic multi-resource generalized assignment problem (DMRGAP), *Eur. J. Oper. Res.*, 105, 91, 1998.

[34] Park, J. S., Lim, B. H., and Lee, Y., A Lagrangian dual-based branch-and-bound algorithm for the generalized multi-assignment problem, *Manage. Sci.*, 44, 271, 1998.

[35] Privault, C. and Herault, L., Solving a real world assignment problem with a metaheuristic, *J. Heuristics*, 4, 383, 1998.

[36] Cattrysse, D. G. and van Wassenhove, L. N., A survey of algorithms for the generalized assignment problem, *Eur. J. Oper. Res.*, 60, 260, 1992.

[37] Yagiura, M., Yamaguchi, T., and Ibaraki, T., A variable depth search algorithm with branching search for the generalized assignment problem, *Optimization Meth. Software*, 10, 419, 1998.

[38] Yagiura, M., Yamaguchi, T., and Ibaraki, T., A variable depth search algorithm for the generalized assignment problem, in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Voß, S., Martello, S., Osman, I. H., and Roucairol, C., Eds., Kluwer Academic Publishers, Boston, MA, 1999, p. 459.

[39] Fisher, M. L., The Lagrangian relaxation method for solving integer programming problems, *Manage. Sci.*, 27, 1, 1981.

[40] Held, M. and Karp, R. M., The traveling salesman problem and minimum spanning trees: Part II, *Math. Prog.*, 1, 6, 1971.

[41] Martello, S., Pisinger, D., and Toth, P., Dynamic programming and strong bounds for the 0-1 knapsack problem, *Management Sci.*, 45, 414, 1999.

[42] Haddadi, S., Lagrangian decomposition based heuristic for the generalized assignment problem, *INFOR*, 37, 392, 1999.

[43] Lorena, L. A. N. and Narciso, M. G., Relaxation heuristics for a generalized assignment problem, *Eur. J. Oper. Res.*, 91, 600, 1996.

[44] Narciso, M. G. and Lorena, L. A. N., Lagrangian/surrogate relaxation for generalized assignment problems, *Eur. J. Oper. Res.*, 114, 165, 1999.

[45] Glover, F., Ejection chains, reference structures and alternating path methods for traveling salesman problems, *Discrete Appl. Math.*, 65, 223, 1996.

[46] Nauss, R. M., The generalized assignment problem, in *Integer Programming: Theory and Practice*, Karlof, J. K., Ed., CRC Press, Boca Raton, FL, 2006, p. 39.

[47] Nonobe, K. and Ibaraki, T., A tabu search approach to the CSP (constraint satisfaction problem) as a general problem solver, *Eur. J. Oper. Res.*, 106, 599, 1998.

[48] Lin, J.-H. and Vitter, J. S., ε-approximations with minimum packing constraint violation, *Proc. STOC,* 1992, p. 771.

[49] Shmoys, D. B. and Tardos, É., An approximation algorithm for the generalized assignment problem, *Math. Prog.*, 62, 461, 1993.

[50] Lenstra, J. K., Shmoys, D. B., and Tardos, É., Approximation algorithms for scheduling unrelated parallel machines, *Math. Prog.*, 46, 259, 1990.

49

Probabilistic Greedy Heuristics for Satisfiability Problems

Rajeev Kohli
Columbia University

Ramesh Krishnamurti
Simon Fraser University

49.1 Introduction

Consider a set of clauses, each clause a disjunction of Boolean variables. The maximum (minimum) satisfiability problem is to find a truth assignment that maximizes (minimizes) the number of clauses that are satisfied. For brevity, we refer to the maximization problem as maxsat, and the minimization problem as minsat. Both maxsat and minsat are NP-hard problems [1–2].

The purpose of this chapter is to examine probabilistic greedy heuristics for the maxsat and minsat problems, and to describe for the simplest of these heuristics the bounds on the average number of clauses satisfied, relative to an optimal solution. We call the algorithms probabilistic greedy heuristics because they set a truth variable true or false not with certainty, but with a probability that depends on the number of unsatisfied clauses in which the variable appears in negated or unnegated form. There are many methods by which the probabilities can be obtained. We discuss some of these, and give the known results for the simplest kinds of rules, in which the probability is (directly or inversely) proportional to the number of additional clauses satisfied if a variable were set true.

There are three main advantages that probabilistic algorithms have over deterministic ones. First, one can often obtain bounds on the average performance of such algorithms without having to specify the method by which the problem instances are generated. In such cases, one is assured of a performance guarantee across all families of statistical distributions giving rise to instances of a problem. This is a form of robustness, obtained by probabilistic algorithms because they can use the structure of a specific problem instance in such a manner as to nullify, at least in part, those peculiarities of data as give rise to worst-case instances for deterministic algorithms. One still obtains worst-case bounds for probabilistic algorithms, but these are bounds on the average performance, which are often higher than the worst-case bounds of similar deterministic algorithms. The second advantage is that analyzing the average performance of probabilistic algorithms is often much simpler than analyzing the average performance of deterministic algorithms. The third advantage is that by selecting the best solution value over a large number of runs of a probabilistic algorithm, one obtains a solution that almost certainly has a worst-case bound that is no smaller than the average performance bound for the probabilistic algorithm. Note that this point is distinct from the possible derandomization of a probabilistic algorithm, as for example the procedure by Mahajan and Ramesh [3] that derandomizes the probabilistic algorithm for maxsat problem by Goemans and Williamson [4].

49.2 Maxsat and Minsat Problems

Let $U = \{u_1, \ldots, u_n\}$ denote a set of n Boolean variables, where each variable $u \in U$ can be true or false. We denote by $\bar{u}$ the negation of a truth variable u. We use the term "literal" to refer to a truth variable in negated or unnegated form. Thus, u and $\bar{u}$ are both literals. A truth assignment for an instance of a satisfiability problem (either maxsat or minsat) comprises a truth setting for each variable, where each variable is set to either true or false.

Let $C = \{c_1, \ldots, c_m\}$ denote a set of m clauses. Each clause $c \in C$ is a disjunction of some, possibly all, of the literals in U. A clause is satisfied if at least one of the literals it contains is true. For example, $c = u_1 \vee \bar{u}_4$, a disjunction of two literals, is satisfied if u_1 is true and/or u_4 is false.

The satisfiability problem is to determine if there is a truth assignment that satisfies all clauses in C. The problem is NP-complete if each clause contains three or more literals [5]. An optimization problem corresponding to satisfiability is the maximum satisfiability or maxsat problem. It is NP-hard [6], and it requires the identification of a truth assignment that satisfies the maximum number of clauses in C. The complementary problem, called minimum satisfiability, or minsat, requires the identification of a truth assignment that satisfies the minimum number of clauses in C. We note that minsat is equivalent to maximizing the number of conjunctive clauses, each of which is satisfied only if all its literals are true.

Minsat is readily solved if there is an assignment that satisfies no clause, because in this case each variable, or its negation, does not appear in any clause. However, the general minsat problem is NP-hard if each clause contains at least two literals [2]. A deterministic approximation algorithm has since been provided for minsat by using an approximation-preserving transformation from minsat to the vertex cover problem [7]. Furthermore, an approximation algorithm with a performance ratio of ρ for minsat implies the existence of an approximation algorithm with the same performance ratio for vertex cover. This suggests that it is hard to provide an approximation algorithm for minsat with a performance ratio better than 2.

Let $u_1, u_2, \ldots, u_n$ denote an arbitrary ordering of the n truth variables in U. We consider the following probabilistic greedy heuristic(s) for solving maxsat and minsat.

Initialization (step 1)

Let $C_1 = C$ denote the set of all clauses in an instance of the maxsat or minsat problem. Let $C_1(u_1)$ denote the subset of clauses in C_1 that contain variable u_1. Let $C_1(\bar{u}_1)$ denote the subset of clauses in C_1 that contain variable $\bar{u}_1$. Let x_1 and y_1 denote the number of clauses in sets $C_1(u_1)$ and $C_1(\bar{u}_1)$. Set

$$u_1 = \begin{cases} \text{True} & \text{with probability } p_1 = f(x_1, y_1) \\ \text{False} & \text{with probability } 1 - p_1 \end{cases}$$

where $f(x_1, y_1) \in [0,1]$. Eliminate all satisfied clauses. Define C_2, the set of clauses not satisfied at the end of step 1:

$$C_2 = \begin{cases} C_1 \setminus C_1(u_1) & \text{if } u_1 \text{is selected at step 1} \\ C_1 \setminus C_1(\bar{u}_1) & \text{if } \bar{u}_1 \text{is selected at step 1} \end{cases}$$

Iteration (step j)

Let C_j denote the set of clauses that are not satisfied at the end of step $j-1$. Let $C_j(u_j)$ denote the subset of clauses in C_j that contain u_j. Let $C_j(\bar{u}_j)$ denote the subset of clauses in C_j that contain $\bar{u}_j$. Let x_j and y_j denote the number of clauses in $C_j(u_j)$ and $C_j(\bar{u}_j)$. Set

$$u_j = \begin{cases} \text{True} & \text{with probability } p_j = f(x_j, y_j) \\ \text{False} & \text{with probability } 1 - p_j \end{cases}$$

where $f(x_j, y_j) \in [0,1]$. Eliminate all satisfied clauses. Define C_{j+1}, the set of clauses not satisfied at the end of step 1:

$$C_{j+1} = \begin{cases} C_j \setminus C_j(u_j) & \text{if } u_j \text{is selected at step } j \\ C_j \setminus C_j(\bar{u}_j) & \text{if } \bar{u}_j \text{is selected at step } j \end{cases}$$

Termination
Stop if $C_{j+1} = \phi$ or if $j = n$.

There are many possible functions $f(\cdot)$ that can be used to determine the probabilities for each of the maxsat and minsat problems. The only conditions are that the value of the function increases with x_j (y_j) and decreases with y_j (x_j) for the maxsat (minsat) problem, and that the probabilities remain bounded between 0 and 1. For example, we can use the function

$$f(x_j, y_j) = \frac{e^{\beta x_j}}{e^{\beta x_j} + e^{\beta y_j}} = \frac{1}{1 + e^{\beta(y_j - x_j)}}$$

for the maxsat problem. The function has value 1/2 when $x_j = y_j$. Its value approaches 1 as x_j becomes much larger than y_j, and approaches zero as y_j becomes much larger than x_j. This is a "logit" function, which has found much use in the statistics and economics literatures [8]. The corresponding function for the minsat problem is

$$f(x_j, y_j) = \frac{e^{\beta y_j}}{e^{\beta x_j} + e^{\beta y_j}} = \frac{1}{1 + e^{\beta(x_j - y_j)}}$$

Perhaps the simplest class of functions we can use is

$$f(x_j, y_j) = \frac{x_j^\beta}{x_j^\beta + y_j^\beta} = \frac{1}{1 + (y_j/x_j)^\beta}, \quad \beta \geq 1$$

for the maxsat problem and

$$f(x_j, y_j) = \frac{y_j^\beta}{x_j^\beta + y_j^\beta} = \frac{1}{1 + (x_j/y_j)^\beta}, \quad \beta \geq 1$$

for the minsat problem. In both cases, $p_j = f(x_j, y_j) = 1/2$ if $x_j = y_j$, and p_j approaches the limiting probabilities 1 (0) in the desired manner. Different values of β give different rates at which the probabilities approach their limiting values; as β becomes arbitrarily large, the probabilities approach the limiting values 0 and 1, and the algorithm becomes a deterministic greedy heuristic. In this sense, a probabilistic rule of the above sort is a generalization of a deterministic greedy heuristic. The only results so far obtained are for the special case of $\beta = 1$ in the above expression; theoretical analysis of the general class of probabilistic greedy heuristics remains open. In the next section, we describe the results that are known for $\beta = 1$ in the above expression.

49.2.1 Performance Bound for Maxsat Problem

In the following discussion, x_j denotes the number of clauses in which the literal u_j occurs, and y_j denotes the number of clauses in which the literal $\bar{u}_j$ occurs. Let $n_j = x_j + y_j$. As noted above, we consider a probabilistic greedy heuristic that sets the jth variable true with probability $p_j = x_j/n_j$.

Without loss of generality, let u_j, $j = 1, \ldots, n$, comprise the optimal solution. Let z_j denote the value of the optimal solution to the subproblem comprising clauses in set C_j. Let a_j denote the value of the optimal solution to the subproblem obtained by eliminating u_j, and all the clauses satisfied when u_j is true. Similarly, let $\bar{a}_j$ denote the value of the optimal solution to the subproblem obtained by eliminating $\bar{u}_j$, and all the clauses satisfied when u_j is false. The following lemma provides a bound on both a_j and $\bar{a}_j$ in terms of z_j, x_j, and n_j.

Lemma 49.1 **(Kohli and Krishnamurti [9])**
$a_j = z_j - x_j$ and $\bar{a}_j \geq max\ \{0, z_j - n_j\}$, for all $j = 1, \ldots, k$.

Proof
Since u_j occurs in x_j clauses, the optimal solution to the maxsat subproblem defined over the set of clauses $C_j \backslash C_j(u_j)$ is, trivially, $a_j = z_j - x_j$. Also, $x_j \leq n_j$ (by definition), so that $z_j - x_j \geq z_j - n_j$.

Of the $z_j - x_j$ clauses in set $C_j \setminus C_j(u_j)$, at most $n_j - x_j$ clauses contain literal $\bar{u}_j$. Hence, the maxsat subproblem defined over the set of clauses $C_j \setminus C_j(\bar{u}_j)$ has an optimal solution $\bar{a}_j$ no smaller than $z_j - x_j - (n_j - x_j) = z_j - n_j$. As n_j can exceed z_j, and as the value of the optimal solution to the maxsat subproblem comprising clauses $C_j \setminus C_j(\bar{u}_j)$ is nonnegative, $\bar{a}_j \geq \max\{0, z_j - n_j\}$, $j = 1, \ldots, k$. □

We now obtain the main result for the maxsat problem.

Theorem 49.1 (**Kohli and Krishnamurti [9]**)

On average, the greedy heuristic for the maxsat problem satisfies at least 2/3 of the number of clauses satisfied by an optimal truth assignment.

Proof

We prove the theorem by induction on the number of variables. We first prove the result for $n = 1$. Without loss of generality, assume that variable u_1 is true in an optimal assignment. Then the value of the optimal solution is $z_1 = x_1$. The expected value of the greedy solution is

$$p_1 x_1 + (1 - p_1)(n_1 - x_1)$$

where $p_1 = x_1/n_1$. Thus, the expected performance ratio of the heuristic is

$$E[r_1] = \frac{1}{z_1}(p_1 x_1 + (1 - p_1)(n_1 - x_1)) = \frac{1}{x_1}\left(\frac{x_1}{n_1}x_1 + \frac{n_1 - x_1}{n_1}(n_1 - x_1)\right)$$

Given n_1, the lower bound on $E[r_1]$ is obtained by minimizing the above expression with respect to x_1, which can be verified to occcur at $x_1 = n_1/\sqrt{2}$. Substituting this value of x_1 in $E[r_1]$ and simplifying yields

$$E[r_1] \geq 2\sqrt{2} - 2 \geq \frac{2}{3}$$

Now suppose

$$E[r_l] \geq \frac{2}{3} \text{ for all } l \leq k - 1$$

We show that

$$E[r_k] \geq \frac{2}{3} \text{ for all } k$$

Suppose the probabilistic greedy heuristic sets u_k true, satisfying x_k clauses. Then the expected number of clauses satisfied by the probabilistic greedy heuristic is no smaller than

$$x_k + \frac{2}{3}a_k$$

By a similar argument, if the greedy heuristic sets u_k false at step 1, the expected value of its solution is no less than

$$n_k - x_k + \frac{2}{3}\bar{a}_k$$

As u_k is selected with probability $p_k = x_k/n_k$, and $\bar{u}_k$ is selected with probability $1 - p_k$, the expected value of the heuristic solution has the lower bound

$$E[f_k] \geq \frac{x_k}{n_k}\left(x_k + \frac{2}{3}a_k\right) + \frac{n_k - x_k}{n_k}\left(n_k - x_k + \frac{2}{3}\bar{a}_k\right)$$

From Lemma 49.1, $a_k \geq z_k - x_k$, which gives

$$E[f_k] \geq \frac{x_k}{n_k}\left(x_k + \frac{2}{3}\left(z_k - x_k\right)\right) + \frac{n_k - x_k}{n_k}\left(n_k - x_k + \frac{2}{3}\bar{a}_k\right)$$

Also, from Lemma 49.1

$$\bar{a}_k \geq \max\{0, z_k - n_k\}$$

Consider $z_k > n_k$. Then

$$\bar{a}_k \geq z_k - n_k > 0$$

and the above inequality for $E[f_k]$ becomes

$$E[f_k] \geq \frac{x_k}{n_k}\left(\frac{2z_k}{3} + \frac{x_k}{3}\right) + \frac{(n_k - x_k)^2}{n_k} + \frac{2(n_k - x_k)}{3n_k}(z_k - n_k)$$

Simplifying

$$E[f_k] \geq \frac{(x_k)^2}{3n_k} + \frac{2z_k x_k}{3n_k} + \frac{(n_k - x_k)^2}{n_k} + \frac{2(n_k - x_k)}{3n_k}(z_k - n_k)$$

The right-hand side obtains its minimum value when $x_k = n_k/2$, which implies

$$E[f_k] \geq \frac{2z_k}{3} \text{ and } E[r_k] = \frac{E[f_k]}{z_k} \geq \frac{2}{3}$$

Now consider $z_k \leq n_k$. Then

$$\bar{a}_k \geq 0\ (\geq z_k - n_k)$$

and therefore

$$E[f_k] \geq \frac{x_k}{n_k}\left(x_k + \frac{2}{3}\left(z_k - x_k\right)\right) + \frac{n_k - x_k}{n_k}(n_k - x_k)$$

Simplifying

$$E[f_k] \geq \frac{x_k^2}{3n_k} + \frac{2z_k x_k}{3n_k} + \frac{(n_k - x_k)^2}{n_k}$$

The right-hand side of the above expression can be verified to obtain its minimum value when

$$x_k = \frac{3n_k - z_k}{4}$$

at which value of x_k

$$E[f_k] \geq \frac{n_k}{4} + \frac{z_k}{2} - \frac{z_k^2}{12n_k}$$

The right-hand side of the above expression takes its smallest value when $n_k = z_k$, for which

$$E[f_k] \geq \frac{z_k}{4} + \frac{z_k}{2} - \frac{z_k}{12} = \frac{2}{3}z_k$$

It follows that

$$E[r_k] = \frac{E[f_k]}{z_k} \geq \frac{2}{3}$$

□

To see that the bound in Theorem 49.1 is tight, consider the following problem instance in which there are k variables and 2^k clauses.

Clause	$u_k\ \bar{u}_k$	$u_{k-1}\ \bar{u}_{k-1}$	. .	$u_2\ \bar{u}_2$	$u_1\ \bar{u}_1$
1	0 1	0 1	. .	0 1	0 1
2	0 1	0 1	. .	1 0	0 0
3	0 1	0 1	. .	0 0	0 0
.	. .	. .	. .	. .	. .
.	. .	. .	. .	. .	. .
2^{k-2}	0 1	0 1	. .	0 0	0 0
$2^{k-2}+1$	0 1	1 0	. .	0 0	0 0
.	. .	. .	. .	. .	. .
.	. .	. .	. .	. .	. .
.	. .	. .	. .	. .	. .
2^{k-1}	0 1	1 0	. .	0 0	0 0
$2^{k-1}+1$	1 0	0 0	. .	0 0	0 0
.	. .	. .	. .	. .	. .
.	. .	. .	. .	. .	. .
.	. .	. .	. .	. .	. .
2^k	1 0	0 0	. .	0 0	0 0

The expected performance of the probabilistic greedy heuristic is

$$\begin{aligned} E[f_k] &= 2\left(\frac{1}{2}\right)\frac{n_k}{2} + 2\left(\frac{1}{2}\right)^2\frac{n_k}{2^2} + \cdots + 2\left(\frac{1}{2}\right)^k\frac{n_k}{2^k} + \left(\frac{1}{2}\right)^k\frac{n_k}{2^k} \\ &= \frac{2n_k}{3} + \frac{n_k}{3}\left(\frac{1}{4^k}\right) \end{aligned}$$

As $z_k = n_k$, the expected performance ratio equals

$$E[r_k] = \frac{2}{3} + \epsilon \text{ where } \epsilon = \frac{1}{3}\cdot\frac{1}{4^k}$$

As ϵ can be made to approach 0 arbitrarily closely by increasing k, $E[r_k]$ can be made to approach 2/3 from above arbitrarily closely. Since the asymptotic upper bound for the probabilistic greedy heuristic is 2/3, the lower bound in Theorem 49.1 is tight.

49.2.2 Performance Bound for Minsat Problem

As in the last section, we restrict our analysis to a probabilistic greedy heuristic for the minsat problem to the case where $\beta = 1$; that is, the heuristic sets the jth variable true with probability $p_j = y_j/n_j$. The following theorem gives the lower bound on the expected performance ratio of the greedy heuristic.

Theorem 49.2 **(Kohli et al. [2])**

On average, the greedy heuristic for the minsat problem satisfies at most twice the number of clauses satisfied by an optimal truth assignment.

Proof

We prove the theorem by induction on the number of variables n. If $n = 1$, the expected number of satisfied clauses is

$$p_1x_1 + (1-p_1)y_1 = \frac{y_1}{n_1}x_1 + \frac{x_1}{n_1}y_1 = \frac{2x_1y_1}{n_1}$$

where $p_1 = y_1/n_1$. Without loss of generality, assume that variable u_1 is true in an optimal assignment. Then the value of the optimal solution is $z = x_1$. Thus, the value of the expected performance ratio for the probabilistic greedy heuristic is

$$E(r_n) = \frac{2x_1y_1}{n_1} = \frac{2y_1}{n_1} \le 2$$

Let $l \ge 1$ be an integer such that

$$E(r_n) \le 2 \quad \text{for } n = l$$

We now show that

$$E(r_n) \le 2 \quad \text{for } n = l + 1$$

If the probabilistic greedy heuristic selects u_1 at step 1, the value of the optimal solution at the second step of the greedy heuristic is $z - x_1$, where z is the optimal solution value of the minsat problem with n variables. However, if the greedy heuristic selects $\bar{u}_1$ at step 1, the value of the optimal solution at the second step is bounded from above by z. Hence, the expected number of clauses satisfied by the probabilistic greedy heuristic is bounded from above by

$$p_1(x_1 + E(r_l)(z - x_1)) + (1 - p_1)(y_1 + E(r_l)z)$$

As $E(r_l) \le 2$ by the induction hypothesis, the value of the above expression is no greater than

$$p_1(x_1 + 2(z - x_1)) + (1 - p_1)(y_1 + 2z)$$

Thus, an upper bound on the expected performance ratio for the probabilistic greedy heuristic is

$$\begin{aligned} E(r_{l+1}) &\le \frac{1}{z}(p_1(x_1 + 2(z - x_1)) + (1 - p_1)(y_1 + 2z)) \\ &= \frac{1}{z}(2z - p_1(x_1 + y_1) + y_1) \\ &\le 2 \end{aligned}$$

□

To prove the above bound is tight, consider the following example with $n = 2$ variables and m clauses. Let

$$\begin{aligned} c_1 &= u_1 \vee u_2 \\ c_2 &= \bar{u}_1 \\ c_i &= \bar{u}_2, \ 3 \le i \le m \end{aligned}$$

The optimal assignment sets both u_1 and u_2 true and satisfies one clause, c_1. The probabilistic greedy heuristic sets u_1 true or false with the same probability (which equals 1/2) at its first step. If it sets u_1 true, then it obtains the optimal solution, setting u_2 true with probability 1 at its second step. Otherwise, at the second step, it sets u_2 true with probability $1 - (1/(m - 1))$, satisfying two clauses, c_1 and c_2; and sets u_2 false with probability $1/(m - 1)$, satisfying $(m - 1)$ clauses, $c_2, \dots, c_m$. Thus, the expected performance ratio (which equals the expected number of satisfied clauses) for the probabilistic greedy heuristic is

$$\frac{1}{2}1 + \frac{1}{2}\left(\frac{m-2}{m-1}(2) + \frac{1}{m-1}(m-1)\right) = 2 - \frac{1}{m-1}$$

As m tends to infinity, the value of this expression approaches from below the bound derived in Theorem 49.2. Note that if we interchange u_2 and $\bar{u}_2$ in the above example, then each clause has at most one unnegated truth variable. Such clauses are called Horn clauses, and so it follows that the above bound on the average performance of the probabilistic greedy heuristic remains tight if we restrict the problem instances to Horn clauses. Also note that $n = 2$ in this example and that the optimal clause c_1 contains

$s = 2$ variables. Thus, the bound on the average performance of the probabilistic greedy heuristic does not depend on m or s.

49.3 Using Probabilistic Algorithms

An average performance analysis does not tell us how well a probabilistic algorithm will do if we stop it after a single run for a given problem instance. It is possible that the solution obtained will be far removed from the average bound. Why then should we be interested in the average performance of a probabilistic algorithm? The answer is that we do not need to stop after a single run. If we run the algorithm a large number of times (say N), then the average of the solution values will asymptotically converge to a value no smaller than the bound on the average performance in Theorems 49.1 or 49.2. The maximum value across these N runs can of course be no smaller than their average. We can thus select this best solution and be assured that, with a very high probability, it has a value that is no worse than the bound on the average performance of the algorithm. The value of N does not have to be enormously large—in most instances, several thousand, if not several hundred runs, will suffice.

There is another way in which probabilistic algorithms can be used. Suppose we run the algorithm a large number of times (say N), and select the best solution. We can repeat the process a large number of times (say M). We then have a sample of M solution values, each the best solution among N solutions. If we plot the distribution of these M best solutions, we obtain an empirical distribution of the best solution. This is a useful distribution, from which we can make several important inferences. First, we note that as N and M become arbitrarily large, the distribution approaches the "true" distribution of the best solution across NM runs. This argument was first used by Fisher and Tippet [10] to obtain closed-form expressions for distributions of a maximum or minimum value from a sample when it is bounded on one or both sides. The main assumption for these distributions to hold is that the samples be drawn from a continuous distribution. This is not valid for such discrete problems as satisfiability, and so the exact distributions need not hold. But the essence of the argument still remains, and one can use repeated runs to obtain a reference distribution which is asymptotically the entire distribution of interest. This procedure is akin to bootstrapping methods in statistics, except that the sampling process uses a probabilistic algorithm. One can use the reference distribution to make probabilistic statements about a solution value, that is, obtain estimates for the probability that the best solution across many runs will exceed a certain numerical value. Such analyses are not possible for deterministic algorithms; but they are natural for probabilistic algorithms, although we are not aware of literature reporting their use.

49.4 Conclusion

There are other probabilistic algorithms for the maxsat problem. The simplest of these is a "random" algorithm that sets each variable true or false with probability 1/2. It is easy to see that the average performance of this algorithm is 1/2 [9]. Johnson's [11] (unweighted) deterministic greedy algorithm with a performance ratio of 1/2 can be considered as a derandomization of this simple random algorithm [12]. A randomized algorithm for maxsat using linear programming rounding, where the fractional value assigned to the variable corresponding to a truth variable is used as the probability of setting the truth variable true, provides a performance guarantee of $1 - 1/e$ [13]. The better of the "random" algorithm and the randomized rounding algorithm using linear programming provides a performance guarantee of 3/4. Approaches using semidefinite programming for the maxsat problem have yielded improved approximation ratios [4]. However, algorithms using linear programming and semidefinite programming have high running times. One benefit of using simple probabilistic algorithms is that the running time is low, and the probabilistic algorithm may be run many times on a given problem instance. By retaining the best solution across all the runs, one obtains a solution that, with high probability, is no worse than the bound on the average performance of a probabilistic algorithm.

Several possible problems remain open, including the following. How do different probability rules affect the expected performance of probabilistic greedy heuristics? Is there an optimal choice of the β parameter for the two rules considered in the paper? Are there some types of problems for which one or another probabilistic greedy heuristic does better than others? And are there ways in which resampling methods like bootstrapping [14] can be used to construct reference distributions from which one can make probabilistic assertions about an obtained solution relative to an optimal solution?

References

[1] Garey, M. R. and Johnson, D. S., *Computers and Intractibility: A Guide to the Theory of NP-Completeness,* W. H. Freeman San Francisco, CA, 1979.

[2] Kohli, R., Krishnamurti, R., and Mirchandani, P., The minimum satisfiability problem, *SIAM J. Disc. Math.,* 7, 275, 1994.

[3] Mahajan, S. and Ramesh, H., Derandomizing approximation algorithms based on semidefinite programming, *SIAM J. Comput.,* 28(5), 1641, 1999.

[4] Goemans, M. X. and Williamson, D. P., Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *JACM,* 42, 1115, 1995.

[5] Even, S., Itai, A., and Shamir, A., On the complexity of timetable and multicommodity flow problems, *SIAM J. Comput.,* 5, 691, 1976.

[6] Garey, M. R., Johnson, D. S., and Stockmeyer, L., Some simplified NP-complete graph problems, *Theor. Comp. Sci.,* 1, 237, 1976.

[7] Marathe, M. V. and Ravi, S. S., On approximation algorithms for the minimum satisfiability problem, *Inf. Proc. Lett.,* 58, 23, 1996.

[8] Maddala, G. S., *Limited-Dependent and Qualitative Variables in Econometrics,* Econometric Society Monographs, Cambridge University Press, New York, 1999.

[9] Kohli, R. and Krishnamurti, R., Average performance of heuristics for satisfiability, *SIAM J. Disc. Math.,* 2, 508, 1989.

[10] Fisher, R. A. and Tippet, L. H. C., Limiting forms of the frequency distribution of the largest or smallest member of a sample, *Proc. Cambridge Philosophical Society,* 24, 1928, p. 180.

[11] Johnson, D. S., Approximation algorithms for combinatorial problems, *JCSS,* 9, 256, 1974.

[12] Vazirani, V. V., *Approximation Algorithms,* Springer, Berlin, 2003.

[13] Goemans, M. X. and Williamson, D. P., New 3/4-approximation algorithms for the maximum satisfiability problem, *SIAM J. Disc. Math.,* 7, 656, 1994.

[14] Efron, B., Computers and the theory of statistics: thinking the unthinkable, *SIAM Rev.,* 21, 460, 1979.

V

Computational Geometry and Graph Applications

50

Approximation Algorithms for Some Optimal 2D and 3D Triangulations

Stanley P. Y. Fung
University of Leicester

Cao-An Wang
Memorial University of Newfoundland

Francis Y. L. Chin
The University of Hong Kong

50.1 Introduction

A *triangulation* [1,2] is a partition of a d-dimensional polytope into simplices. The decomposed simplices should be nonoverlapping except sharing the common vertices (0-faces), edges (1-faces), faces (2-faces), . . . , $(d-1)$-faces. When $d = 2$, a triangulation is the subdivision of a two-dimensional (2D) region into empty triangles, in which two adjacent triangles share an edge. When $d = 3$, a triangulation, sometimes called *tetrahedralization*, can be viewed as a decomposition of a three-dimensional (3D) volume into a set of disjoint 3D empty tetrahedra in which two adjacent tetrahedra share a face. Triangulation is not only an interesting theoretical problem in computational geometry, it also has many important applications, such as finite element methods [3] for computer-aided design and physical simulations.

In this chapter we consider some approximation algorithms for triangulations in two and three dimensions. Note that a 2D region (or a 3D volume) can be defined by a simple polygon (or a polyhedron), or by a set of points, in which case we are to triangulate the region enclosed by the convex hull of the set of points. We consider both the cases of simple polygon/polyhedron and point sets. Throughout this chapter, S denotes a set of points, P denotes a polygon or polyhedron, n denotes the number of points in S or the number of vertices in P, and $CH(S)$ denotes the convex hull of S.

A standard assumption in computational geometry is to assume the points in consideration are in *general position*. In our case, we assume no three points are collinear and no four points are on the same circle in the 2D case, and no four points are on the same plane in the 3D case. In particular, for a convex polyhedron in 3D, this implies that all faces of the polyhedron are triangles.

In the rest of this section we will present some properties and simple heuristics of 2D and 3D triangulations. Subsequent sections will discuss some 2D and 3D triangulation approximation algorithms in detail.

50.1.1 2D Triangulations

Let $S = \{s_i \mid i = 1, \ldots, n\}$ be a set of n points in a plane such that S is in general position. Let $s_i s_j$ for $i \neq j$ denote the line segment with endpoints s_i and s_j, and let $|s_i s_j|$ denote the Euclidean distance between s_i and s_j. A *triangulation* of S, denoted by $T(S)$, is a maximum set of noncrossing line segments with their endpoints in S.

A 2D simple polygon or point set may admit more than one triangulation, but all triangulations contain the same number of triangles. An arbitrary triangulation of a simple polygon can be found in linear time [4]. However, there are many objectives which we can optimize and have practical applications, for example, maximizing the minimum (or minimizing the maximum) angle [5], edge length [6], or circumcircle, of the triangles in the triangulation. One well-known triangulation is called the *Delaunay triangulation* (DT) [7,8], which is closely related to Voronoi diagrams. It can be computed by an efficient $O(n \log n)$ time algorithm. It has many interesting properties, such as maximizing the minimum angle of the DT triangles, and that each circumscribing circle of a DT triangle of a set of points contains no other points in its interior. One variation of the DT problem is to have the triangulation be constrained by a given set of line segments, called *Constrained Delaunay triangulation* (CDT) [9,10]. When the set of constrained line segments forms a simple polygon, there is a linear time algorithm for computing the CDT [11].

Greedy triangulations [12] are constructed by inserting the diagonals of the polygon one by one in sorted order according to some parameters as long as the later inserted diagonals do not intersect with those already inserted. A greedy triangulation for a 2D polygon can be constructed in $O(n \log n)$ time.

There is another famous triangulation called *minimum-weight triangulation* (MWT) [13], which has the property that the sum of lengths of all line segments in the triangulation is minimized. Finding the minimum-weight triangulation of a point set is a famous outstanding open problem; it is still unknown whether this problem has a polynomial-time algorithm or is NP-complete [14].[1] Properties on minimum-weight triangulations can be found in Refs. [15–17]. The *maximum-weight triangulation* (MAT) problem [18] is similar to the MWT problem except that the sum of lengths of all line segments in the triangulation is maximized.

We will present in Sections 50.2 and 50.3 a number of heuristics for the MWT and MAT problems.

50.1.2 3D Triangulations

Properties of 3D Triangulations

The 3D triangulation, or tetrahedralization, has many properties different from 2D triangulation. For example, a polygon in 2D can have many different triangulations, where each triangulation contains the same number of triangles. However, different tetrahedralizations of the same polyhedron can result in different numbers of tetrahedra. A simple example is given in Figure 50.1; one tetrahedralization gives two tetrahedra while the other gives three.

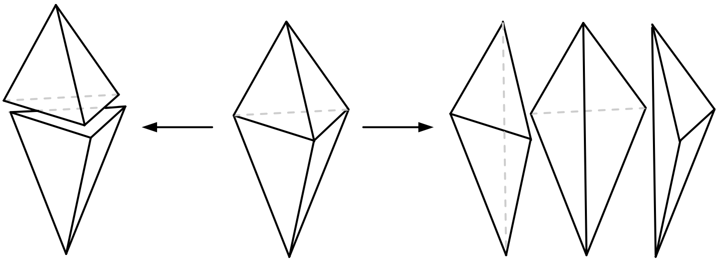


FIGURE 50.1 Two different tetrahedralizations.

[1] Very recently, Mulzer and Rote showed that the minimum weight triangulation problem is NP-hard [68].

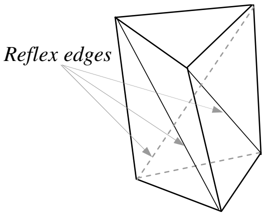


FIGURE 50.2 A nontetrahedralizable example: A twisted triangular prism.

The difference in the number of tetrahedra in two tetrahedralizations can in fact be quite large: consider the n vertices of a polyhedron to be on the moment curve, $\{v_i = (i, i^2, i^3), i = 1, \ldots, n\}$. This polyhedron can be decomposed into $\binom{n-2}{2}$ tetrahedra of the form $v_i v_{i+1} v_j v_{j+1}$. With Euler's formula, this is the maximum number of tetrahedra that a tetrahedralization of an n-vertex polyhedron can have. However, as we shall see, there exists a tetrahedralization for the same polyhedron containing at most $2n-7$ tetrahedra.

Another interesting property is that, unlike 2D simple polygons, triangulation is not even always possible: not all simple polyhedra are tetrahedralizable. A famous example is owed to Schönhardt [19], which is a twisted triangular prism (Figure 50.2). The triangular base of the prism is twisted so that each of the three rectangular faces folds into two triangular faces with a reflex edge between them. The resultant polyhedron is nonconvex and cannot be tetrahedralized without inserting additional vertices inside the polyhedron, called *Steiner points*. In general, it is NP-complete [20] to determine whether a 3D nonconvex polyhedron is tetrahedralizable without Steiner points. As we shall see, Steiner points are not necessary for convex polyhedra. In this chapter we only consider triangulations without Steiner points.

Another interesting difference is the greedy 3D tetrahedralization of a set of points, which, unlike the greedy 2D triangulation, might not always be possible if edges are inserted in some order according to some parameters, such as their lengths. The problem becomes NP-complete if the selected edge has to be checked, besides whether it and its induced component intersect with the previously inserted components, whether its inclusion with the previously inserted edges is tetrahedralizable [21,22].

Some Triangulation Heuristics for Convex Polyhedra

All convex polyhedra can be tetrahedralized by the *pulling* (also known as "*fan-out*") approach, that is, by choosing a particular vertex v in the polyhedron and forming tetrahedra with faces in the polyhedron which are not adjacent to v. Since each face of the polyhedron is a triangle, by Euler's formula, the total number of faces in an n-vertex polyhedron must be exactly $2n-4$. Let d be the degree of v, then the number of nonadjacent faces (or equivalently the number of decomposed tetrahedra) is $2n - 4 - d \leq 2n - 7$ as $d \geq 3$. The pulling tetrahedralization may not result in the minimum number of tetrahedra. An example is given in Figure 50.3: The convex polyhedron is formed by attaching four flat tetrahedra onto the four

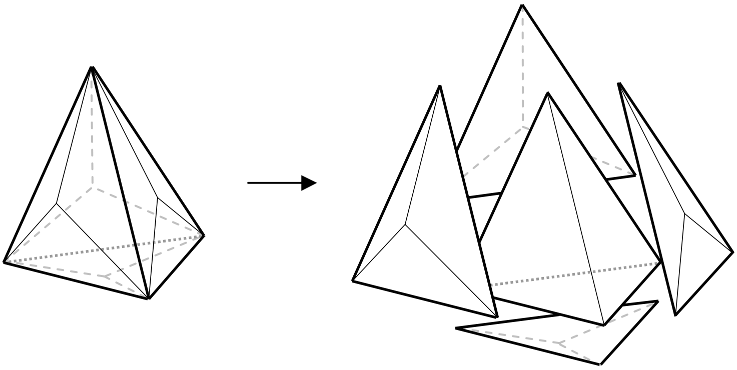


FIGURE 50.3 An optimal tetrahedralization which is not "pulling".

faces of a tetrahedron. The pulling tetrahedralization of this polyhedron gives at least six tetrahedra, while the optimal tetrahedralization gives five.

Another way to tetrahedralize a polyhedron is by *peeling*, also called *cap-removal* or *shelling*. The peeling algorithm is similar to what is given in Ref. [23] for polyhedron with reflex edges. Let $N(v)$ be the set of neighbors of a vertex v. The *cap* of v is defined as the star-shaped polyhedron formed by removing the convex hull of $N(v)$ from the convex hull of $N(v) \cup v$, that is, $\text{cap}(v) = CH(N(v) \cup v) - CH(N(v))$. The cap-removal algorithm removes the caps of the vertices one by one, and each of the removed caps is tetrahedralized by the pulling approach: if v is of degree d, the cap of v can be replaced by $(d-2)$ tetrahedra. This cap-removal step is performed recursively until the whole polyhedron is tetrahedralized. As the degree of v can be $O(n)$, the cap of v is replaced by $O(n)$ tetrahedra. This approach takes at most $O(n^2 \log n)$ time and can produce $O(n^2)$ tetrahedra. As the surface of the polyhedron forms a planar graph, there always exists a vertex of degree at most 5, and we can tetrahedralize the polyhedron by recursively removing the cap of those vertices with degree 5 or less. This cap-removal algorithm based on vertices of minimum degrees produces a tetrahedralization with at most $3n - 11$ tetrahedra; it is another tetrahedralization algorithm which results in a linear number of decomposed tetrahedra other than the pulling approach.

How to tetrahedralize a convex polyhedron with the minimum number of tetrahedra is a well-known open problem [2], until it is shown recently to be NP-hard [24]. Only very special types of polyhedra have known minimum triangulations [25,26]. It is therefore natural to consider approximation algorithms for this problem. In Section 50.4, we give approximation algorithms for finding a tetrahedralization for a convex polyhedron that give the minimum number of tetrahedra. We will also consider nonapproximability of this problem.

50.2 2D Minimum Weight Triangulation

In this section we consider MWTs for a set of points in two dimensions. The weight of a triangulation $T(S)$ is given by

$$\omega(T(S)) = \sum_{s_i s_j \epsilon T(S)} | s_i s_j |$$

An MWT of S, denoted by $MWT(S)$, is defined as a triangulation such that $\omega(MWT(S)) = \min\{\omega(T(S))\}$ for all possible $T(S)$.

One of the outstanding open problems listed in Garey and Johnson's book [14] is MWT. The complexity status of this problem is unknown since 1975 [27]. A great deal of work has been done to seek the ultimate solution of the problem. Basically, there are three directions to approach the problem: (1) to construct the exact $MWT(S)$ for restricted classes of point sets [28–31]. In particular, when the point set forms a simple polygon P with n vertices, then $MWT(P)$ can be computed in $O(n^3)$ time [28,29]; (2) to identify edges inclusive or exclusive to $MWT(S)$ [15,32–35] and hoping that the identified edges may form a constant number of connected components or several simple polygons. Then, a dynamic programming method can solve this problem in polynomial-time; and (3) to find a triangulation $T(S)$ which has a good approximation ratio: $\omega(T(S))/\omega(MWT(S))$. This direction of research has practical significance since finding the MWT may be time-consuming even if solvable in polynomial-time.

In direction (1), Gilbert and Klincsek [28,29] independently showed an $O(n^3)$ time dynamic programming algorithm to obtain $MWT(P)$, where P is restricted to be a simple n-gon. Anagnostou and Corneil [30] gave an $O(n^{3k+1})$ time algorithm to find $MWT(S)$, where S are vertices of k nested convex polygons. Meijer and Rappaport [36] gave an $O(n^k)$ time algorithm when S is restricted to be on k nonintersecting lines. It was shown in Ref. [31] that if given a subgraph of $MWT(S)$ with k connected components, then the complete $MWT(S)$ can be computed in $O(n^{k+2})$ time. A sparse point set case was studied in Ref. [37]. Most recently, Hoffmann and Okamoto [38] proved that $MWT(S)$ is fixed-parameter tractable, that is, $MWT(S)$ can be solved in $O(6^k n^5 \log n)$ time, where k is the number of points of S inside the convex hull $CH(S)$. Thus, when $k \leq O(\log n)$, this time complexity is polynomial. A different algorithm (which also handles the case of k points inside a nonconvex polygon) with time complexity $O(n^3k!k)$ is given in Ref. [39].

In direction (2), researchers noted that the intersection of all possible $T(S)$, called *stable edges* in Ref. [40], is a subgraph of $MWT(S)$. A triangulation is *k-local minimal* if every k-sided simple polygon formed by the edges of this triangulation is an MWT of that k-gon. Since $MWT(S)$ must be k-local minimal (for $k \leq n$), the intersection of all 4-local minimal triangulations of S is a subgraph of $MWT(S)$, called the *LMT-skeleton* [41]. Dickerson and Montague [41] showed an algorithm to compute the LMT-skeleton. Dai and Katoh [42] used a variation of the local minimal method to a version of MWT that restricts the angles of the triangles. Experiments show that the local minimal method usually results in a connected subgraph; however, it was shown [43] that there exist point sets such that a connected subgraph cannot be formed using these local minimal methods.

Researchers also noted that edges with certain local properties must belong to $MWT(S)$. Gilbert [28] showed that the shortest edge in S is in $MWT(S)$. Keil [15] proved that the so-called β-skeleton of S for $\beta = \sqrt{2}$ is a subgraph of $MWT(S)$, which inspired a wave of research in this direction. Cheng and Xu [34] extended Keil's result to $\beta = 1.17682$. Yang et al. [33] showed that mutual nearest neighbors are also in $MWT(S)$. Wang et al. [16] investigated the case of nonsymmetric geometric condition for an edge in $MWT(S)$. Das and Joseph [32] identified some edges which do not belong to any $MWT(S)$. Drysdale et al. [44] identified exclusion regions of $MWT(S)$. The edge identification of $MWT(S)$ seems to be a promising approach and has the following merit: the more edges of $MWT(S)$ being identified, the fewer number of disconnected components are there in S. Thus, it is possible that eventually all these identified edges form a connected graph so that an $MWT(S)$ can be constructed by dynamic programming in polynomial time [31]. So far, all these MWT edges identified were *light edges*, where an edge is called a light edge if it is shorter than all other edges crossing it. It is still open whether or not some nonlight edges can be identified in polynomial-time and they together with the known light edges form a connected graph.

In direction (3), researchers concentrated on designing efficient triangulation algorithms with good approximation ratios with respect to $MWT(S)$. It has been shown that the well-known greedy triangulation and DT do not approximate $MWT(S)$ well: Kirkpatrick [45] showed that DT may have approximation factor $\Omega(n)$ with respect to $MWT(S)$, and Levcopoulos [46] demonstrated that greedy triangulation may have an approximation factor of $\Omega(\sqrt{n})$.

To see this, refer to Figure 50.4(a), where $n - 1$ vertices lie on a circle with radius r and within a very small arc of length δ, where $\delta \ll r$, and one vertex lies on the center of the circle. The DT connects the center vertex with all other vertices and the total length is $r(n - 1) + \delta$. However, MWT of this set would first connect all $n - 1$ vertices on the arc to form a triangulation, and then connect the center vertex with

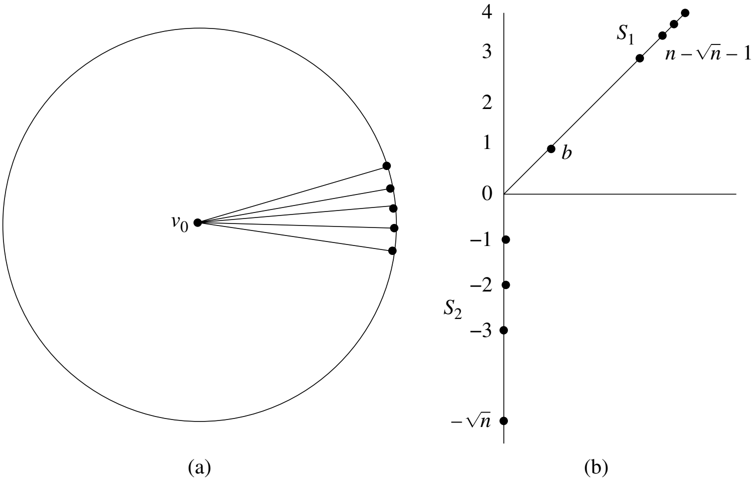


FIGURE 50.4 Delaunay and greedy triangulations may be a bad approximation.

the two extreme vertices on the arc. Thus, the total length of MWT would be at most $2r + (2n - 5)\delta$. Thus, we have an $\Omega(n)$ factor for DT. (By slight modifications, we can avoid this degenerated case and still obtain the $\Omega(n)$ factor.)

For the greedy triangulation, refer to Figure 50.4(b), where $\sqrt{n}$ points (S_2) are put evenly on the negative y-axis, one point (called blocking point, b) is put on coordinates (1, 1), and the remaining $(n - \sqrt{n} - 1)$ points (S_1) are on the line between coordinates (3,3) and (4,4). The greedy triangulation $GT(S)$ contains edges connecting vertex b to all $\sqrt{n}$ vertices in S_2 and the edges connecting vertex $(0, -\sqrt{n})$ to all vertices in S_1. The total length of $GT(S)$ is at least $|CH(S)| + \sum_{i=1}^{\sqrt{n}} \sqrt{1 + (i+1)^2} + (n - \sqrt{n} - 1) \cdot \sqrt{3^2 + (\sqrt{n} + 3)^2} = \Theta(n\sqrt{n})$, where $|CH(S)|$ is the length of edges on the convex hull of S. But $MWT(S)$ contains edges of $CH(S)$, edges connecting vertex $(0, -1)$ to all the vertices in S_1 as well as b, and the edges connecting vertex (4,4) to all the vertices in S_2. The total length of $MWT(S)$ is at most $|CH(S)| + \sum_{i=1}^{\sqrt{n}} \sqrt{4^2 + (i+4)^2} + (n - \sqrt{n} - 1)\sqrt{5^2 + 4^2} + \sqrt{5} = O(n)$. Thus, the ratio $\omega(GT(S))/\omega(MWT(S)) = \Omega(\sqrt{n})$.

Lingas [47] and Heath and Pemmaraju [48] took another approach. First, build a minimum spanning tree on the Delaunay or the greedy triangulation, which partitions the point set into a number of simple polygons. Then use dynamic programming to find the optimal triangulation of each of these smaller polygons. This heuristic is no worse than the simple Delaunay or greedy triangulation, respectively, but not better either: they only yield $\Omega(n)$ and $\Omega(\sqrt{n})$ approximation ratio, respectively [49]. It is proved in Ref. [50] that the simple greedy triangulation or the one based on building a minimum spanning tree first both have approximation ratio $O(\sqrt{n})$, so the bound is tight.

Before the above $O(\sqrt{n})$ bound on greedy triangulation was proved, Plaisted and Hong [51] already proposed an $O(\log n)$ factor approximation algorithm. The best approximation algorithm is owed Levcopoulos and Krznaric [50]. They provided a constant factor approximation algorithm, even though the constant is very large.

Finally, we turn our attention to convex polygons. Even though the MWT of a simple n-gon can be found in $O(n^3)$ time, no better algorithm has been found. Researchers turned to design some efficient approximation algorithms for this case. Levcopoulos and Lingas [52] proved that the greedy triangulation yields a constant factor approximation to the MWT of a convex polygon. Levcopoulos and Krznaric gave a linear-time approximation scheme [53], that is, it has approximation ratio $1 + \epsilon$ for arbitrarily small $\epsilon > 0$.

We will in the following briefly discuss Plaisted and Hong's $O(\log n)$ factor algorithm, Levcopoulos and Krznaric's constant factor algorithm, and Lingas and Levcopoulos's constant factor greedy triangulation for convex polygons.

50.2.1 An $O(\log n)$ Approximation Algorithm for Point Sets

This approximation algorithm is based on two steps:

1. Partition the point set S into a set of empty convex polygons, that is, the interior of each polygon does not contain any point in S. The total lengths of the boundaries of these convex polygons are proportional to $\omega(MWT(S))$.
2. Use the so-called "ring heuristic" to triangulate each of the empty convex polygons.

We now explain the steps in more detail. The algorithm first finds the "initial star" for each vertex inside the convex hull of S, where an initial star for a vertex p is a group of three edges incident to p such that the angle between two adjacent edges is less than 180° and the "initial star" has the smallest total edge length among all possible such stars at p. For example in Figure 50.5(a), p_4 has star $(p_5 p_8 p_9)$, p_5 has star $(p_2 p_6 p_9)$, p_6 has star $(p_3 p_5 p_9)$, p_7 has star $(p_1 p_3 p_4)$, p_8 has star $(p_1 p_4 p_5)$, and p_9 has star $(p_4 p_6 p_8)$. The total edge lengths of all initial stars are bounded by $2\omega(MWT(S))$. This is because in any inner vertex, say p, there exists a star of $MWT(S)$ incident to p, and the length of this star is at least the length of p's initial star. Since each edge in the initial stars may be counted twice, we have a factor of 2.

However, some edges of the initial stars may cross. To form a set of convex polygons, one needs to replace the crossing edges by one or more noncrossing edges. For example in Figure 50.5(a), edge $p_4 p_7$ crosses

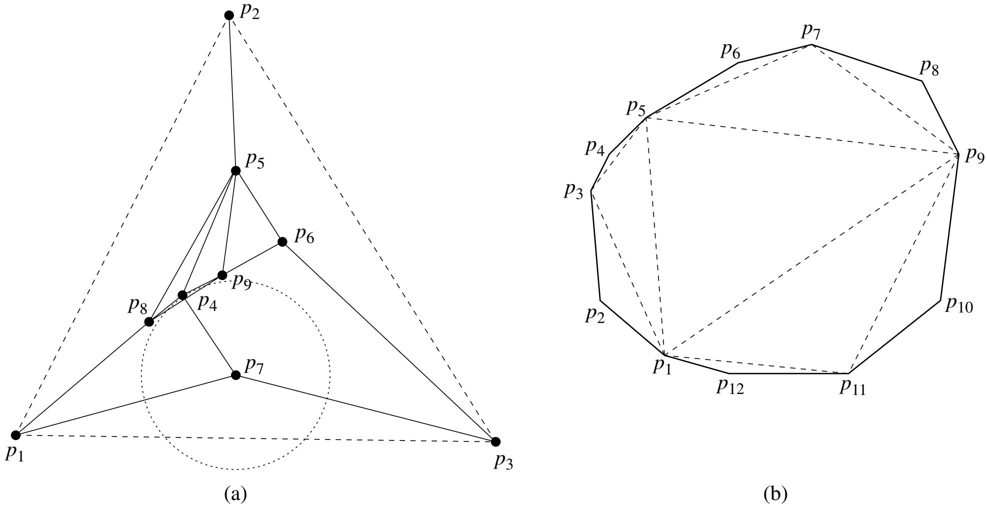


FIGURE 50.5 (a) An illustration for the stars. (b) The ring heuristic.

edge $p_8 p_9$. In the process, $p_4 p_7$ may be replaced by, for example, $p_7 p_9$. The algorithm processes the set of initial star edges, S_1, in ascending order of their lengths, and attempts to add them to the modified edge set S_2 which is initially empty. If an edge $p_i p_j$ in S_1 does not cross any edge in S_2, it is moved to S_2. Otherwise, $p_i p_j$ is removed from S_1, and either one or more edges are added into S_2 or an edge is added to a "tentative" edge set according to a 14-case analysis. The process ensures that these newly added edges have total length at most twice the length of $p_i p_j$, and the edges in S_2 do not cross each other. Furthermore, if a tentative edge is shorter than twice the length of $p_i p_j$, a special treatment is applied to replace this edge with some edges which are then moved to S_2.

To complete the triangulation, the algorithm uses the "ring heuristic" to triangulate each convex polygon resulted in the first step. To do so, the algorithm first connects every other vertex by an edge, that is, connecting $p_1 p_3$, $p_3 p_5$, $\ldots$, $p_{n-1} p_1$ (assuming $n = 2^k$ for integer k) to form a ring of edges. Then it connects $p_1 p_5$, $p_5 p_9, \ldots$, $p_{n-3} p_1$ to form another inner ring of edges, and continues in this manner until only three vertices are left. Figure 50.5(b) shows the ring heuristic triangulation of a convex polygon with 12 vertices. It is obvious that the total length of the edges in each inner level of ring is less than the total length of an outer ring. Since there are about $\lceil \log n \rceil$ rings, the total length of the edges inside the convex polygon is at most $O(\log n)$ times the length of the boundary of the polygon.

Thus, this approximation algorithm yields a triangulation with $O(\log n)$-factor to the $MWT(S)$.

50.2.2 A Constant Factor Approximation Algorithm for Point Sets

It was noted that the greedy triangulation and DT can approximate MWT to a constant factor when the points are uniformly distributed [54,55]. Levcopoulos and Krznaric [50] showed that only in one certain distribution, the greedy triangulation approximates MWT badly. This case occurs when the point set forms two concave chains facing each other (see Figure 50.6). To avoid the failure of the greedy algorithm, they modified the greedy triangulation to check this case.

Specifically, let G be a graph initially containing only the vertex set S. Let E be the edge set sorted in nondecreasing order of edge lengths. The algorithm tries to insert the shortest edge currently in E into G (like greedy triangulation). Let $v_1 u_1$ be an edge to be inserted. If all the following six conditions hold, then an edge $v_0 u_0$ is added into G, otherwise the edge $v_1 u_1$ is added into G.

1. The diagonal $v_1 u_1$ forms an empty triangle (v_1, u_0, u_1) with two edges in G.
2. There is a diagonal $v_0 u_0$ properly intersecting $v_1 u_1$ and forming an empty triangle (v_0, v_1, u_0) with two edges in G.

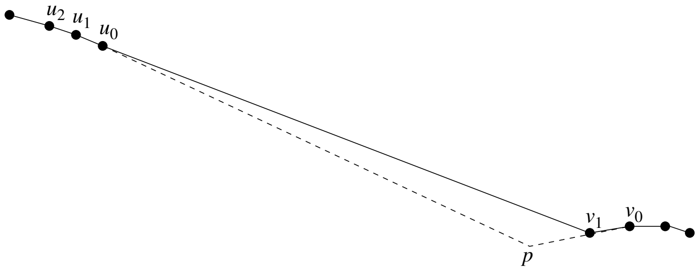


FIGURE 50.6 An illustration for the quasi-greedy algorithm.

3. The angle $\angle(v_1, u_0, u_1) > 135°$ in triangle (v_1, u_0, u_1).
4. $|u_0v_0| < 1.1|u_1v_1|$.
5. $|v_0p| < 0.5|u_0p|$, where p is the intersecting point of the straight-line extensions of $|v_0v_1|$ and $|u_0u_1|$.
6. There is an edge u_1, u_2 in G such that (v_1, u_0, u_1, u_2) forms an empty quadrangle and the angle $\angle(u_0, u_1, u_2)$ in that quadrangle is greater than 180°.

Levcopoulos and Krznaric proved [50] that this modified greedy algorithm yields a triangulation of a point set with a constant factor of the MWT. The detailed proof is omitted here.

50.2.3 A Linear-Time Approximation Algorithm for Convex Polygons

When the given point set forms a simple polygon, its MWT can be found in $O(n^3)$ time by dynamic programming. However, it is still open that whether or not there exists a sub-$O(n^3)$ algorithm for even a convex polygon, and an $O(n^3)$ time algorithm is still too expensive in practice. Hence, researchers tried to design efficient algorithms to approximate $MWT(P)$ for a convex polygon P.

Levcopoulos and Lingas [52] proved that greedy triangulation $GT(P)$ of a convex polygon can approximate MWT within a constant factor. Their proof basically consists of the following parts. (1) In a convex polygon P, if a diagonal d intersects and hence partitions a greedy edge e of $GT(P)$ into two parts e' and e'', then either $|e'| = O(|d|)$ or $|e''| = O(|d|)$. A similar property holds for MWT edges. Therefore, the length of an edge of $GT(P)$ is related to the length of an edge of $MWT(P)$ if they cross each other. (2) For each edge d in $MWT(P)$, the set of greedy triangulation edges that intersects d and within certain weight ranges has constant cardinality. (3) The total length of all edges of $GT(P)$ incident to a vertex v_i is only a constant factor larger than the length of the longest edge of $GT(P)$ incident to v_i.

They used the above properties and case analysis to match the edges in $MWT(P)$ to edges in $GT(P)$ and proved that $\omega(GT(P)) = O(\omega(MWT(P))$.

50.3 2D Maximum Weight Triangulation

Unlike MWT, there is not much research done on $MAT(S)$ of a point set S. $MAT(S)$ is defined as a triangulation such that $\omega(MAT(S)) = \max\{\omega(T(S))\}$ for all possible $T(S)$. From the theoretical viewpoint, the MAT and MWT problems should be of equal interest, and one seems not to be easier than the other.

The first work in MAT showed that if an n-sided polygon P is inscribed on a circle, then $MAT(P)$ can be found in $O(n^2)$ time instead of $O(n^3)$ [18]. Hu [56] proved that greedy, greedy spanning tree, and maximum spanning tree triangulation heuristics all have approximation ratio not better than $\Omega(\sqrt{n})$. Then, Hu gave a simple "spoke triangulation" approximation algorithm to triangulate a point set S with an approximation ratio of 6. Chin et al. [57] improved Hu's result to 4.238 by presenting a nontrivial upper bound of the weight of $MAT(S)$ for a set S of n points in the plane. If the point set forms a convex polygon,

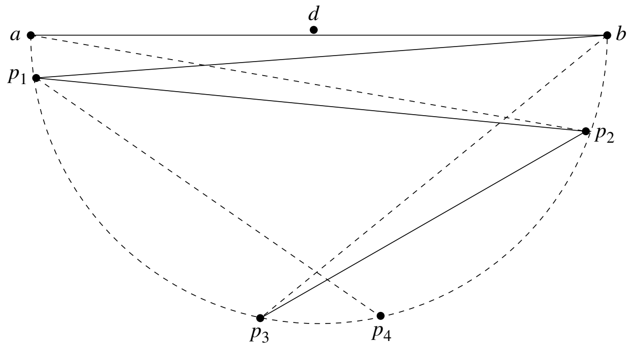


FIGURE 50.7 An illustration for the $\Omega(\sqrt{n})$ ratio.

the spoke triangulation algorithm gives an approximation ratio of 2 [56], and Qian and Wang [58] gave a linear-time approximation scheme that gives an approximation ratio of $1 + \epsilon$ for any $\epsilon > 0$. In the following, we will briefly discuss each of the above results for point sets.

Greedy Triangulation Approximates MAT Badly

To see why, consider a point set as in Figure 50.7. All points are put on or inside a unit half circle with diameter ab, except the point d which is slightly above ab. The first $\sqrt{n}$ points $p_1, p_2, \ldots$ are placed in such a manner that the greedy triangulation will select the edges bp_1, p_1p_2, $p_2p_3, \ldots$. The total length of these edges in $GT(S)$ can be shown to be $O(\sqrt{n})$. Then the rest of $n - \sqrt{n}$ points are put within a small triangle $p_{\sqrt{n}-3}p_{\sqrt{n}-2}p_{\sqrt{n}-1}$ and again their total edge length in $GT(S)$ is at most $O(\sqrt{n})$. Therefore, $\omega(GT(S)) = O(\sqrt{n})$. On the other hand, an $MAT(S)$ would have weight $\Omega(n)$ just by connecting point d to all other points in S and adding $CH(S)$. Thus, $\frac{\omega(MAT(S))}{\omega(GT(S))} = \Omega(\sqrt{n})$.

The Spoke Triangulation Algorithm

The algorithm can be described as follows. Let $D_S = |s_is_j|$ be the diameter of a point set S. The line extending D_S partitions S into S^1 and S^2 (excluding s_i and s_j). Let $\omega(E_i^1)$ and $\omega(E_j^1)$ (respectively, $\omega(E_i^2)$ and $\omega(E_j^2)$) denote the sum of the lengths of edges connecting every point in S^1 (respectively, S^2) to s_i and s_j, respectively. The algorithm first takes the "spokes" in the larger one of $\omega(E_i^1)$ and $\omega(E_j^1)$ (respectively, the larger one of $\omega(E_i^2)$ and $\omega(E_j^2)$) as the first subset of the triangulation edges. Then, it completes the triangulation similar to Graham's scan convex hull construction algorithm. It was shown that this triangulation has weight at least $(n+1)\frac{D_S}{2}$. Since there are at most $3n-6$ edges in any triangulation of n points, this gives a trivial bound of $\omega(MAT(S)) \leq (3n-6)D_S$ and therefore the algorithm has an approximation ratio of 6.

Improving the Approximation Ratio to 4.238

Chin et al. [57] observed that not all $3n-6$ edges can be as long as the diameter. They proved a tighter upper bound and hence improved the approximation ratio to 4.238. The intuitive idea of proof is as follows.

Let ϵ be a constant, $0 < \epsilon \leq \frac{1}{2}$. We call a triangle $\triangle$ "large" if its perimeter $\omega(\triangle) \geq (2 + 2\epsilon) \cdot D_\triangle$ where $D_\triangle$ is the diameter of the triangle, and "small" otherwise. Note that ϵ is the shape parameter of a triangle: a triangle with perimeter $(2 + 2\epsilon)D_\triangle$ is long and thin if $\epsilon \to 0$ and equilateral if $\epsilon = 1/2$.

To maximize the edge lengths of a triangulation of the set, one may wish that a triangulation contains as many long and thin ("large") triangles as possible. However, it can be proved that if a triangle $\triangle$ has $\omega(\triangle) \geq (2+2\epsilon)D_\triangle$, then its area $A_\triangle \geq \epsilon\sqrt{1-\epsilon^2} \cdot D_\triangle^2$. Since the maximum area of the convex hull of a point set with diameter D_S is bounded above by $\frac{\pi}{4}D_S^2$, we have that $T(S)$ may contain at most $\frac{\pi}{4\epsilon\sqrt{1-\epsilon^2}}$ large triangles. It is well known that any triangulation $T(S)$ contains at most $2n-5$ triangles. Suppose $T(S)$ contains m large triangles and at most $2n - 5 - m$ small triangles. It is clear that the perimeter of a large triangle is at most $3D_S$, and the perimeter of a small one is less than $(2+2\epsilon) \cdot D_S$ by definition. Summing up the perimeters of all the triangles in $T(S)$ as well as the perimeter of $CH(S)$ (which is at most πD_S), we obtain a sum equals twice of $\omega(T)$. Simplifying, we have $\omega(T) \leq ((2+2\epsilon)\cdot n + \frac{\pi(1-2\epsilon)}{8\epsilon\sqrt{1-\epsilon^2}} + \frac{\pi}{2} - 5(1+\epsilon))D_S$.

If we apply the spoke-triangulation algorithm to produce a triangulation $T'(S)$ of a point set, then by our upper bound on $\omega(T(S))$, we obtain an approximation ratio

$$\frac{\omega(T(S))}{\omega(T'(S))} \le \frac{((2+2\epsilon)\cdot n + \frac{\pi(1-2\epsilon)}{8\epsilon\sqrt{1-\epsilon^2}} + \frac{\pi}{2} - 5(1+\epsilon))D_S}{(n+1)D_S/2} = (4+4\epsilon) + \frac{b(\epsilon)}{n+1}$$

where $b(\epsilon)$ is some function of ϵ. By setting $\epsilon = 0.05932$, the approximation ratio is always at most $4 + 4\epsilon = 4.238$.

50.4 Minimum Triangulation of 3D Convex Polyhedra

As noted in the introduction, finding the minimum tetrahedralization of a 3D convex polyhedron is NP-hard. In this section we consider the problem of finding an approximation to the minimum-size triangulation of a convex polyhedron. For a convex polyhedron P, let n denote its number of vertices and Δ denote its maximum vertex degree.

To bound the approximation ratio of an algorithm, we need to have a bound on the size of the minimum (i.e., optimal) triangulation. A triangulation may contain *interior edges*, which are edges of the set of decomposed tetrahedra not on the surface of the polyhedron. For any triangulation of an n-vertex polyhedron, the number of interior edges e is directly related to the number of tetrahedra t by the formula $t = e + n - 3$, which was shown in Refs. [59–61]. In the following this will be used as a lower bound on the size of optimal triangulations. Note that this implies a lower bound of $n - 3$ tetrahedra for any triangulation.

50.4.1 Simple Heuristics

In Section 50.1.2 we introduced simple triangulation heuristics like pulling and peeling. The pulling heuristic gives a triangulation of size $2n - 4 - d$, where d is the degree of the vertex where we "pull" from. To minimize the size of triangulation, the pulling should be done from the maximum-degree vertex. So the minimum triangulation has size at most $2n - 4 - \Delta$. Recall that there is a lower bound of $n - 3$ for the size of any triangulation. Hence pulling gives an approximation ratio of $(2n - 4 - \Delta)/(n - 3) = 2 - \Theta(1/n)$. Similarly, since peeling gives a triangulation of size at most $3n - 11$, it gives an approximation ratio of $3 - \Theta(1/n)$.

50.4.2 A Better Approximation Algorithm

We now describe another triangulation algorithm [62], which gives a slightly better approximation ratio $2 - \Omega(1/\sqrt{n})$ by exploiting the combinatorial properties of polyhedra and their triangulations.

A 3-*cycle* in the (surface) graph of a polyhedron is a closed path of three edges in the graph such that each side of the cycle contains at least one other vertex. In other words, a 3-cycle partitions the polyhedron into two convex subpolyhedra (the requirement of having at least one other vertex on each side ensures the 3-cycle is not a face of the polyhedron, which does not partition the polyhedron).

The new algorithm, CutPull, adds a simple extra step comparing with pulling: It partitions the polyhedron along all 3-cycles before applying pulling.

- *Step 1.* Find all 3-cycles on the surface graph of the given polyhedron P. This can be done efficiently using algorithms such as those in Refs. [63,64].
- *Step 2.* Partition the polyhedron along all these 3-cycles to produce a set of subpolyhedra, each is free of 3-cycles.
- *Step 3.* Apply the pulling heuristic to each of the resulting subpolyhedron.

Suppose there are k 3-cycles in a polyhedron P. We have the following three properties concerning 3-cycles and CutPull, whose proofs are elementary (see Ref. [62]).

1. Let P_1, P_2 be the resulting subpolyhedra when P is partitioned along a 3-cycle. Let n_1 and n_2 be the number of vertices of P_1 and P_2, and t, t_1, and t_2 be the number of tetrahedra produced by the pulling heuristic applied to P, P_1, and P_2, respectively. Then $t \geq t_1 + t_2$. (In other words, cutting along 3-cycles does not make pulling worse.)
2. CutPull produces a triangulation with at most $2n - 7 - k$ tetrahedra.
3. If P has maximum vertex degree Δ and $n > 4$, and has a minimum triangulation with e_m interior edges, then $e_m\Delta + 3k \geq n - 2$.

To prove the approximation ratio of CutPull, consider a convex polyhedron P, and suppose it has k 3-cycles. Consider the following two cases:

Case 1
$k = o(n)$. Since $e_m\Delta \geq n - 2 - 3k = n - o(n)$, and using the arithmetic-geometric-mean inequality, this gives $e_m + \Delta \geq 2\sqrt{n - o(n)} = \Theta(\sqrt{n})$. So

$$\frac{2n-4-\Delta}{e_m+n-3} \leq \frac{2n-4-(\Theta(\sqrt{n})-e_m)}{e_m+n-3} \leq \frac{2n-4-\Theta(\sqrt{n})}{n-3} \leq 2 - \Omega(\frac{1}{\sqrt{n}})$$

This means even if the pulling heuristic is applied directly to P (without first cutting 3-cycles), we still have an approximation ratio of $2 - \Omega(\frac{1}{\sqrt{n}})$. Since partitioning the polyhedron along all 3-cycles before pulling will not increase the number of tetrahedra, CutPull also gives an approximation ratio of $2 - \Omega(\frac{1}{\sqrt{n}})$.

Case 2
$k = \Theta(n)$. The approximation ratio is then at most $\frac{2n-7-cn}{e_m+n-3} \leq 2 - c < 2 - \Omega(\frac{1}{\sqrt{n}})$ for some constant $c > 0$.

Therefore, CutPull gives an approximation ratio of $2 - \Omega(\frac{1}{\sqrt{n}})$ in either case.

50.4.3 Lower Bound for Approximation Ratio

No general lower bound on the approximation ratio of this problem is known. It is known, however, that if we only consider the *combinatorial structure* of the polyhedron, the bound of CutPull in Section 50.4.2 is actually tight. Two polyhedra have the combinatorial structure if their graph is isomorphic. Polyhedra with the same combinatorial structure can still have their vertices at different relative positions, and a different minimum size in triangulations (see Ref. [60]). Note that CutPull and pulling considers only combinatorial structures.

To prove the lower bound we need the following observation. If two polyhedra, A and B have the same combinatorial structure, and their sizes of minimum triangulations are t_A and t_B respectively, where $t_A < t_B$, then no algorithm that only considers combinatorial structures can have approximation ratio better than t_B/t_A. This is because the algorithm cannot distinguish A and B, and can only produce the same output for them, which must be at least t_B. The main idea of the proof is to construct two convex polyhedra P1 and P2 such that they have the same combinatorial structure, but the sizes of their minimum triangulations differ by a large amount.

We do not go into the details of the constructions; they can be found in Ref. [62]. Figure 50.8 shows a rough idea of the construction of a polyhedron from which P1 and P2 will be formed. It consists of m wedges $W_1, \ldots, W_m$. Wedge W_k has vertices a_k, b_k, c_k, d_k. In each wedge W_k, m vertices $q_k^1, q_k^2, \ldots, q_k^m$ are created between each interval (c_k, d_k) and edges $a_kq_k^i$, $b_kq_k^i$ are added. The polyhedron is formed by taking the convex hull of all wedge vertices, and has $n = m^2 + 4m$ vertices. The wedges are carefully placed, and do not intersect one another except touching at a single point. P1 and P2 are formed by slight changes to this polyhedron. For P1, the wedges are "pushed" slightly together so that they all "interlock" each other. For P2, the wedges are "moved away" a little bit so that they do not intersect.

P1 and P2 have the same combinatorial structure, but their sizes of minimum triangulations greatly differ, because the size of triangulations of the wedges differ greatly depending on whether they are interlocked.

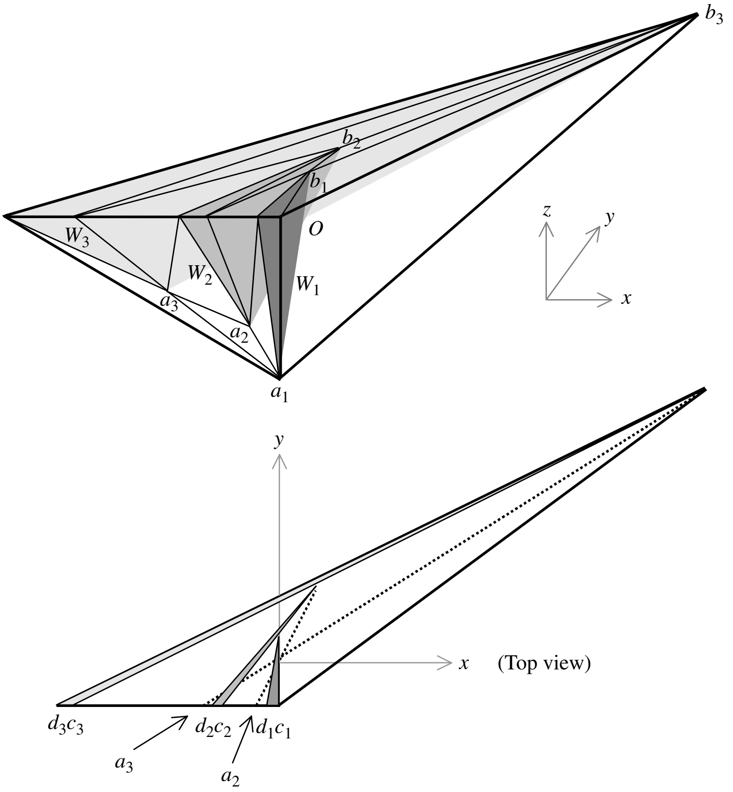


FIGURE 50.8 The placement of three wedges; q^i's are not shown.

In P1, all wedges are interlocked, and a total of at least $2m(m-1)+(m+1) = 2n - \Theta(\sqrt{n})$ tetrahedra must appear in any triangulation. For P2, each wedge can be triangulated independently, producing $m+1$ tetrahedra each. The remaining space can also be triangulated using $O(m)$ tetrahedra. Thus there is a triangulation with total number of tetrahedra $m(m+1)+O(m) = n+\Theta(\sqrt{n})$.

It follows that any approximation algorithm for finding minimum triangulations of convex polyhedra, using only combinatorial structures, cannot have approximation ratio better than $2 - O(\frac{1}{\sqrt{n}})$.

Note that unlike most other nonapproximability results for NP-hard problems, this result does not depend on the P/NP question: This lower bound for approximation is valid no matter P = NP or not. It is because the bound is derived by observing the ambiguity in the size of triangulations due to combinatorial restrictions. However, this lower bound proof may not hold if noncombinatorial information can be used.

50.4.4 Special Cases

The CutPull algorithm gives better approximation ratios for some special classes of polyhedra [65], which we briefly discuss below.

(1) *Knowledge of maximum degree.* The CutPull algorithm actually gives an approximation ratio of $2 - \Omega(\frac{1}{\Delta}) - \Omega(\frac{\Delta}{n})$ where Δ is the maximum degree. This is proved along the lines of the general case $2 - \Omega(\frac{1}{\sqrt{n}})$ bound in Section 50.4.2. Thus, for example the approximation ratio is $2 - c$ for some constant c if $\Delta = \Theta(n)$. The worst case happens when $\Delta = \Theta(\sqrt{n})$ in which the bound reduces to $2 - \Omega(\frac{1}{\sqrt{n}})$. Similar to Section 50.4.3, a matching lower bound for the case where only combinatorial structures are considered can be shown.

(2) *Constant maximum degree.* When the maximum degree Δ is constant, the bound in (1) above can be shown to be at most $2-\frac{2}{\Delta+1}-\Omega(\frac{1}{n})$. This bound improves substantially from 2, especially when Δ is small. For example, when $\Delta = 6$, the approximation ratio $r \leq 12/7 \approx 1.714$; when $\Delta = 7$, $r \leq 7/4 = 1.75$.

(3) *No 3-cycles and no degree-4 vertices.* If we can prove a better lower bound on the size of minimum triangulations for special types of polyhedra, a better approximation ratio can be obtained using the same algorithm. We have seen that 3-cycles and wedges (which consist of long chains of degree-4 vertices) makes approximating the size of minimum triangulation difficult. If we consider the special type of polyhedra without these two structures, a lower bound of $(4n - 8)/3$ on the size of any minimum triangulation can be proved. Therefore, the CutPull algorithm gives an improved approximation ratio of 3/2 in this case. This bound is tight in the sense that there are polyhedra attaining this approximation ratio using CutPull.

(4) *Minimum degree at least 5.* This is a generalization of the above type but allowing 3-cycles. If a polyhedron has no degree-3 and degree-4 vertices and k 3-cycles, it can be shown that (using similar arguments as in [3] above) the size of any minimum triangulation is at least $(4n-8)/3-4k$. Since CutPull gives a triangulation of size at most $2n-7-k$, the approximation ratio is $\frac{2n-7-k}{\max(n-3,(4n-8)/3-4k)}$. This ratio is less than $2-\frac{1}{12} = 1.9166\ldots$ for any k.

50.5 Conclusion and Open Problems

This chapter studies some triangulation problems for point sets and polygons in two and three dimensions. The 2D triangulation problems are widely studied and constant factor approximation algorithms (for point sets) and linear-time approximation schemes (for convex polygons) are known, but the complexity of finding the exact optimal triangulation remains unknown. 3D triangulations are relatively less studied and not much is known other than minimizing the size of triangulations.

Some more important open problems include:

- What is the complexity of finding MWT of a 2D point set? Is it solvable in polynomial-time or is NP-complete?
- Are there better approximation algorithms for finding MWTs of a 2D point set?
- Are there subcubic time algorithms for finding MWTs of convex polygons?
- For 3D polyhedron triangulation, are there algorithms with better approximation ratios ($r < 2-\epsilon$ where $\epsilon > 0$ is constant, independent from n) when the actual coordinatizations are taken into account, instead of just looking at the combinatorial structure?
- Are there other kinds of polyhedra that have better approximation ratios (or can be triangulated optimally) using CutPull or other algorithms?
- What about other objective functions in three dimensions? For example, O'Rourke [66] raises the problem of maximum-size tetrahedralizations and points out some motivations. There are also work on minimizing the total area of the triangles [67].

References

[1] Bagemihl, F., On indecomposable polyhedra, *Am. Math. Mon.*, 55(7), 411, 1948.

[2] Bern, M. and Eppstein, D., Mesh generation and optimal triangulation, in *Computing in Euclidean Geometry*, 2nd ed., Du, D.-Z. and Hwang, F. K., Eds., World Scientific, Singapore, 1995, p. 47.

[3] Castillo, J. E., Ed., Mathematical aspects of numerical grid generation, *SIAM,* 1991.

[4] Chazelle, B., Triangulating a simple polygon in linear time, *Disc. and Comput. Geom.*, 6, 485, 1991.

[5] Edelsbrunner, H., Tan, T. S., and Waupotitsch, R., A polynomial time algorithm for the minmax angle triangulation, *SIAM J. Sci. Stat. Comp.*, 13, 994, 1992.

[6] Edelsbrunner, H. and Tan, T. S., A quadratic time algorithm for the minmax length triangulation, *Proc. of FOCS,* 1991, p. 414.

[7] Fortune, S., Voronoi diagrams and Delaunay triangulations, in *Computing in Euclidean Geometry*, Hwang, F. K. and Du, D.-Z., Eds., World Scientific, Singapore, 1992.

[8] Aurenhammer, F., Voronoi diagrams—a survey of a fundamental geometric data structure, *ACM Comput. Surv.*, 23, 345, 1991.

[9] Chew, L. P., Constrained Delaunay triangulations, *Algorithmica*, 4, 97, 1989.

[10] Wang, C. and Schubert, L., An optimal algorithm for constructing the Delaunay triangulation of a set of line segments, *Proc. 3rd ACM Symp. on Comput. Geom.*, 1987, p. 223.

[11] Chin, F. and Wang, C. A., Finding the constrained Delaunay triangulation and constrained voronoi diagram of a simple polygon in linear-time, *SIAM J. Comput.*, 28(2), 471, 1998.

[12] Levcopoulos, C. and Lingas, A., Fast algorithms for greedy triangulation. *BIT*, 32, 280, 1992.

[13] Düppe, R. D. and Gottschalk, H. J., Automatische Interpolation von Isolinien bei willkürlichen Stützpunkten, *Allgemeine Vermessungsnachrichten*, 77, 423, 1970.

[14] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[15] Keil, J. M., Computing a subgraph of the minimum weight triangulation, *Comput. Geom.: Theor. Appl.*, 4, 13, 1994.

[16] Wang, C. A., Chin, F. Y. L., and Xu, Y., A new subgraph of minimum weight triangulations, *J. Comb. Opt.*, 1(2), 115, 1997.

[17] Wang, C. A., Chin, F. Y. L., and Yang, B., Triangulations without minimum-weight drawing, *Inf. Proc. Lett.*, 74(5–6), 183, 2000.

[18] Wang, C. A., Chin, F. Y. L., and Yang, B. T., Maximum weight triangulation and graph drawing, *Inf. Proc. Lett.*, 70(1), 17, 1999.

[19] Schönhardt, E., Uber die Zerlegung von Dreieckspolyedern in Tetraeder, *Math. Annalen.,* 98, 309, 1928.

[20] Ruppert, J. and Seidel, R., On the difficulty of tetrahedralizing 3-dimensional non-convex polyhedra, *Disc. Comput. Geom.*, 7, 227, 1992.

[21] Chin, F. and Wang, C. A., On the difficulty of greedy tetrahedralization of points in 3D, *Proc. 7th Canadian Conf. on Comput. Geom.*, 1995, p. 285.

[22] Yang, B., Wang, C. A., and Chin, F. Y. L., Algorithms and complexity for tetrahedralization detections, *Proc. 13th Int. Symp. on Algorithms and Computation*, 2002, p. 296.

[23] Chazelle, B. and Palios, L., Triangulating a nonconvex polytope, *Disc. Comput. Geom.*, 5, 505, 1990.

[24] Below, A., De Loera, J., and Richter-Gebert, J., The complexity of finding small triangulations of convex 3-polytopes, *J. Algorithms*, 50(2), 134, 2004.

[25] Wang, C. A. and Yang, B., Optimal tetrahedralizations of some convex polyhedra, *European Workshop on Comput. Geom.*, 2000, p. 5.

[26] Yang, B. and Wang, C. A., Minimal tetrahedralizations of a class of polyhedra, *J. Comb. Opt.*, 8(3), 241, 2004.

[27] Shamos, M. I. and Hoey, D., Closest point problems, *Proc. of FOCS*, 1975, p. 151.

[28] Gilbert, P., New Results on Planar Triangulations, Technical Report ACT-15, Coord. Sci. Lab., University of Illinois at Urbana, 1979.

[29] Klincsek, G., Minimal triangulations of polygonal domains, *Ann. Discrete Math.*, 9, 121, 1980.

[30] Anagnostou, E. and Corneil, D., Polynomial time instances of the minimum weight triangulation problem, *Comput. Geom.: Theor. Appl.*, 3, 247, 1993.

[31] Cheng, S.-W., Golin, M., and Tsang, J., Expected case analysis of β-skeletons with applications to the construction of minimum weight triangulations, *Proc. 7th Canadian Conf. on Comput. Geom.*, 1995, p. 279.

[32] Das, G. and Joseph, D., Which triangulations approximate the complete graph? *Proc. Int. Symp. on Optimal Algorithms*, Lecture Notes in Computer Science, Vol. 401, Springer, Berlin, 1989, p. 168.

[33] Yang, B. T., Xu, Y. F., and You, Z. Y., A chain decomposition algorithm for the proof of a property on minimum weight triangulations, *Proc. 5th Int. Symp. on Algorithms and Computation*, Lecture Notes in Computer Science, Vol. 834, Springer, Berlin, 1994, p. 423.

[34] Cheng, S.-W. and Xu, Y.-F., Approaching the largest β-skeleton within a minimum weight triangulation, *Proc. 12th ACM Symp. on Comput. Geom.*, 1996, p. 196.

[35] Wang, C. and Yang, B., A lower bound for β-skeleton belonging to minimum weight triangulations, *Comput. Geom.: Theor. and Appl.*, 19(1), 35, 2001.

[36] Meijer, H. and Rappaport, D., Computing the minimum weight triangulation of a set of linearly ordered points, *Inf. Proc. Lett.*, 42(1), 35, 1992.

[37] Wang, C. and Xu, Y., Computing a minimum weight triangulation of a sparse point set, *J. Global Opt.*, 15, 73, 1999.

[38] Hoffmann, M. and Okamoto, Y., The minimum weight triangulation problem with few inner points. *Proc. 1st Int. Workshop on Parameterized and Exact Computation*, Lecture Notes in Computer Science, Vol. 3162, Springer, Berlin, 2004, p. 200.

[39] Grantson, M., Borgelt, C., and Levcopoulos, C., Minimum weight triangulation by cutting out triangles, *Proc. 16th Int. Symp. on Algorithms and Computation*, 2005.

[40] Mirzaian, A., Wang, C., and Xu, Y., On stable line segments in triangulations, *Proc. 8th Canadian Conf. on Comput. Geom.*, 1996, p. 62.

[41] Dickerson, M. T. and Montague, M. H., A (usually?) connected subgraph of the minimum weight triangulation, *Proc. 12th ACM Symp. on Comput. Geom.*, 1996, p. 204.

[42] Dai, Y. and Katoh, N., On computing new classes of optimal triangulations with angular constraints, *Proc. 4th Int. Conf. of Computing and Combinatorics*, Lecture Notes in Computer Science, Vol. 1449, Springer, Berlin, 1998, p. 15.

[43] Bose, P., Devroye, L., and Evans, W., Diamonds are not a minimum weight triangulation's best friend, *Proc. 8th Canadian Conf. on Comput. Geom.*, 1996, p. 68.

[44] Drysdale, R. L., McElfresh, S., and Snoeyink, J., On exclusion regions for optimal triangulations, *Disc. Appl. Math.*, 109 (1–2), 49, 2001.

[45] Kirkpatrick, D. G., A note on Delaunay and optimal triangulations, *Info. Proc. Lett.*, 10(3), 127, 1980.

[46] Levcopoulos, C., An $\Omega(\sqrt{n})$ lower bound for the nonoptimality of the greedy triangulation, *Inf. Proc. Lett.*, 25, 247, 1987.

[47] Lingas, A., A new heuristic for minimum weight triangulation, *SIAM J. Algebraic Disc. Meth.*, 8(4), 646, 1987.

[48] Heath, L. and Pemmaraju, S., New results for the minimum weight triangulation problem, *Algorithmica*, 12(6), 533, 1994.

[49] Levcopoulos, C. and Krznaric, D., Tight lower bounds for minimum weight triangulation heuristics, *Inf. Proc. Lett.*, 57(3), 129, 1996.

[50] Levcopoulos, C. and Krznaric, D., Quasi-greedy triangulations approximating the minimum weight triangulation, *J. Algorithms*, 27(2), 303, 1998.

[51] Plaisted, D. A. and Hong, J., A heuristic triangulation algorithm, *J. Algorithms*, 8(3), 405, 1987.

[52] Levcopoulos, C. and Lingas, A., On approximation behavior of the greedy triangulation for convex polygons, *Algorithmica*, 2, 175, 1987.

[53] Levcopoulos, C. and Krznaric, D., A linear-time approximation scheme for minimum weight triangulation of convex polygons, *Algorithmica*, 21(3), 285, 1998.

[54] Chang, R. C. and Lee, R. C. T., On the average length of Delaunay triangulations, *BIT*, 24(3), 269, 1984.

[55] Levcopoulos, C. and Lingas, A., The greedy triangulation approximates the minimum weight triangulation and can be computed in linear time in the average case, Technical report LU-CS-TR:92-105, Department of Computer Science, Lund University, Sweden, 1992. Preliminary version in *Proc. ICCI'*91, Lecture Notes in Computer Science, p. 497.

[56] Hu, S., A constant-factor approximation for maximum weight triangulation, *Proc. 15th Canadian Conf. on Comput. Geom.*, 2003, p. 150.

[57] Chin, F., Qian, J., and Wang, C., Progress on maximum weight triangulation, *Proc. 10th Int. Conf. on Computing and Combinatorics*, 2004, p. 53.

[58] Qian, J. and Wang, C. A., A linear-time approximation scheme for maximum weight triangulation of convex polygons, *Algorithmica*, 40, 161, 2004.

[59] Edelsbrunner, H., Preparata, F. P., and West, D. B., Tetrahedrizing point sets in three dimensions, *J. Symbolic Comput.*, 10, 335, 1990.

[60] Below, A., Brehm, U., De Loera, J. A., and Richter-Gebert, J., Minimal simplicial dissections and triangulations of convex 3-polytopes, *Disc. Comput. Geom.*, 24(1), 35, 2000.

[61] Richter-Gebert, J., Finding small triangulations of polytope boundaries is hard, *Discrete Comput. Geometry*, 24, 503, 2000.

[62] Chin, F. Y. L., Fung, S. P. Y., and Wang, C. A., Approximation for minimum triangulations of simplicial convex 3-polytopes, *Disc. Comput. Geom.*, 26(4), 499, 2001.

[63] Papadimitriou, C. H. and Yannakakis, M., The clique problem for planar graphs, *Inf. Proc. Lett.*, 13, 131, 1981.

[64] Chiba, N. and Nishizeki, T., Arboricity and subgraph listing algorithms, *SIAM J. Comput.*, 14(1), 210, 1985.

[65] Fung, S. P. Y., Chin, F. Y. L., and Poon, C. K., Approximating the minimum triangulation of convex polytopes with bounded degrees, *Comput. Geom.: Theor. Appl.*, 32(1), 1, 2005.

[66] O'Rourke, J., Computational geometry column 40, *Int. J. Comput. Geom. and Appl.*, 10(6), 649, 2000.

[67] Cheng, S.-W. and Dey, T. K., Approximate minimum weight steiner triangulation in three dimensions, *Proc. of SODA*, 1999, p. 205.

[68] Mulzer, W. and Rote, G., Minimum weight triangulation is NP-hard, *Proc. 22nd Symposium on Computational Geometry*, 2006, pp. 1–10.

51

Approximation Schemes for Minimum-Cost *k*-Connectivity Problems in Geometric Graphs

Artur Czumaj
University of Warwick

Andrzej Lingas
Lund University

51.1 Introduction

51.1.1 Multiconnectivity Problems

We survey the recent progress in the design of approximation schemes for geometric variants of the following classical optimization problem: for a given undirected weighted graph, find its minimum-cost subgraph that satisfies a priori given multiconnectivity requirements. We present the approximation schemes for various geometric minimum-cost *k*-connectivity problems and for geometric survivability problems, giving a detailed tutorial of the novel techniques developed for these algorithms. We also shortly discuss extensions to include planar graphs.

A classical multiconnectivity graph problem is as follows: for a given undirected weighted graph, find its minimum-cost subgraph that satisfies a priori given connectivity requirements.

Multiconnectivity graph problems are central in algorithmic graph theory and have numerous applications in computer science and operation research (see, e.g., Refs. [1–5]). They also play very important role in the design of networks that arise in practical situations (see, e.g., Refs. [1,2,6]). Typical application areas include telecommunication, computer, and road networks. Low-degree connectivity problems for geometrical graphs in the plane can often closely *approximate* such practical connectivity problems (see, e.g., the discussion in Refs. [2,5,7]). For instance, they can be used to model the design of low-cost telephone networks that can "survive" some types of edge and node failure. In such a model, the cost of the edge corresponds to the cost of lying a fiber-optic cable between the endpoints of the edge plus the planned cost of the service of the cable. Furthermore, the minimum connectivity requirement for a pair of vertices corresponds to the minimum number of edge and/or node failures that must occur in the network before the pair is completely disconnected. In practice, the latter value tends to be quite low, usually not more than 2, since failures are assumed to be isolated accidents, such as fires at nodes [5,7]. Note that the cost of lying a fiber-optic cable between two points is roughly proportional to the length of the link (see, e.g., Ref. [7]).

In this work, we survey approximation results for these problems restricted to geometric graphs and planar graphs.

The most classical problem we study is the (*Euclidean*) *minimum-cost k-vertex connected spanning subgraph* (*k-VCSS*) *problem.* We are given a set S of n points in the Euclidean space $\mathbb{R}^d$ and the aim is to find a minimum-cost k-vertex connected Euclidean graph spanning points in S (i.e., a subgraph of the complete Euclidean graph on S).

Throughout the chapter we shall assume that the cost of the graph is equal to the sum of the costs of the edges of the graph. Furthermore, in the geometric case, the cost of an edge connecting a pair of points $x, y \in \mathbb{R}^d$ is equal to the Euclidean distance between points x and y, that is, $\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$, where $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$. More generally, the distance could be defined using other norms, such as ℓ_p norms for any $p > 1$; all results discussed in this survey can be extended from the Euclidean case to other ℓ_p norms.

By substituting the requirement of k-edge connectivity for that of k-vertex connectivity, we obtain the corresponding *(Euclidean) minimum-cost k-edge connected spanning subgraph (k-ECSS) problem.* We term the generalization of the latter problem which allows for parallel edges in the output graph spanning S as the *(Euclidean) minimum-cost k-edge connected spanning sub-multigraph (k-ECSSM) problem.*

The concept of minimum-cost k-connectivity naturally extends to include that of *Euclidean Steiner k-connectivity* by allowing the use of additional vertices, called *Steiner points.* For a given set S of points in $\mathbb{R}^d$, we say that a geometric graph G is a *Steiner k-VCSS* (*or, Steiner k-ECSS*) *for S* if the vertex set of G is a *superset* of S and for every pair of points from S there are k internally vertex-disjoint (edge-disjoint, respectively) paths connecting them in G. The problem of *(Euclidean) minimum-cost Steiner k-vertex-* (*or, k-edge-*) *connectivity* is to find a minimum-cost Steiner k-VCSS (or, Steiner k-ECSS) for S. For $k = 1$, it is simply the *Steiner minimal tree* (SMT) problem, which has been very extensively studied in the literature (see, e.g., Refs. [8,9] and Chapter 42).

In a more general formulation of multiconnectivity graph problems, nonuniform connectivity constraints have to be satisfied. The *survivable network design problem* is defined as follows: for a given weighted undirected graph $G = (V, E)$ and a connectivity requirement function $r : V \times V \rightarrow \mathbb{N}$, find a minimum-cost subgraph of G such that for any pair of vertices $x, y \in V$ the subgraph has $r_{x,y}$ internally vertex-disjoint (or edge-disjoint, respectively) paths between x and y. Also in that case, the output may be allowed to be a multigraph [5]. The survivable network design problem arises in many aforementioned applications, e.g., in telecommunication, communication network design, and VLSI design.

In many applications of this problem, often regarded as the most interesting ones [2,10], the connectivity requirement function is specified with the help of a one-argument function, which assigns to each vertex

v its connectivity type $r_v \in \mathbb{N}$. Then, for any pair of vertices v, $u \in V$, the connectivity requirement $r_{u,v}$ is simply given as $\min\{r_u, r_v\}$ [2,5,7,11,12]. Following the literature, we assume this standard simplification of the connectivity requirements function in this chapter. Note that, in particular, this includes the *Steiner tree problem* (see, e.g., Ref. [13]), in which $r_v \in \{0, 1\}$ for any vertex $v \in V$. It also includes the most widely applied variant of the survivability problem in which $r_v \in \{0, 1, 2\}$ for any vertex $v \in V$ (see, e.g., Ref. [2,5,7]).

Since all the aforementioned k-connectivity problems are known to be $\mathcal{NP}$-hard when restricted to even two-dimensions for $k \geq 2$ [14], we focus on efficient constructions of good approximations. We aim at developing a *polynomial-time approximation scheme* (*PTAS*). This is a family of algorithms $\{\mathcal{A}_\varepsilon\}$ such that, for each fixed $\varepsilon > 0$, $\mathcal{A}_\varepsilon$ runs in time polynomial in the size of the input and produces a $(1 + \varepsilon)$-approximation (see Chapter 9 and Ref. [15]).

51.1.2 History of the Multiconnectivity Problems

For a very extensive presentation of results concerning problems of finding minimum-cost k-vertex- and k-edge-connected spanning subgraphs, nonuniform connectivity, connectivity augmentation problems, and geometric problems, we refer the reader to Refs. [1,4,16] and to various chapters of Ref. [15], especially to Refs. [3,17]. Here we discuss mostly the work related to geometric graphs.

All the multiconnectivity problems discussed in this survey are known to be $\mathcal{NP}$-hard not only for general graphs, but also for several nontrivial classes of graphs. For general graphs, the multiconnectivity problems are even known to be APX-hard, that is, they do not have a PTAS unless $\mathcal{P} = \mathcal{NP}$ (see Ref. [18]). Despite the practical relevance of the multiconnectivity problems for geometrical graphs and the vast amount of practical heuristic results reported (see, e.g., Refs. [2,5,7,12]), very little theoretical research has been done toward developing efficient approximation algorithms for these problems until a few years ago. This contrasts with the very rich and successful theoretical investigations of the corresponding problems in general metric spaces and for general weighted graphs. And so, until 1998, even for the simplest and most fundamental multiconnectivity problem, that of finding a minimum-cost biconnected graph spanning a given set of points in the Euclidean plane, obtaining approximations achieving better than a $\frac{3}{2}$ ratio had been elusive (the ratio $\frac{3}{2}$ is the best polynomial-time approximation ratio known for general graphs whose weights satisfy the triangle inequality [19]; for other results, see e.g., Refs. [17,20]).

For many years the algorithmic community has believed that TSP and the multiconnectivity problems discussed in this survey were also APX-hard for geometric and planar graphs, and hence they do not have a PTAS unless $\mathcal{P} = \mathcal{NP}$. However, the situation changed dramatically with the seminal works of Arora [13] and Mitchell [21], who showed that the TSP problem in geometric graph has a PTAS; soon after, a PTAS for TSP in planar graphs has been also developed by Arora et al. [22]. These results gave a hope that also the multiconnectivity problems in geometric and planar graphs can have a PTAS. This has been proven affirmatively in a series of papers by Czumaj and Lingas [18,23,24] in the case of geometric graphs. The case of planar graphs seems to be more challenging, and as for today, we know only a partial solution for two-connectivity problems, where a quasi-polynomial time approximation scheme (running-time of the form $n^{\widetilde{\mathcal{O}}(\log n/\varepsilon)}$) has been recently developed [25,26].

51.1.3 Overview of the Results

In this survey, we overview the recent polynomial-time approximation schemes for multiconnectivity problems in geometric graphs, as developed in a series of papers by Czumaj et al. in Refs. [18,23,24,27] (see also Refs. [28,29] for full versions of the papers). Besides presenting the specific results, we give a detailed tutorial of techniques developed during the design of polynomial-time approximation schemes for various k-connectivity problems in geometric graphs; we also emphasize the difference between these algorithms and the recent PTASs for TSP and related problems. In addition, we discuss lower bounds on approximability of these problems in higher dimensions [18] and extensions to include the survivability problems [27] and planar graphs [25,26].

51.2 Preliminaries

In this section, we introduce basic technical definitions and notions used in this survey. For simplicity of presentation we shall assume that the quality of approximation ε satisfies $n^{-1/4} < \varepsilon \leq 0.1$. Furthermore, we shall aim at achieving an approximation of $(1 + \mathcal{O}(\varepsilon))$ rather than $(1 + \varepsilon)$. Both these assumptions can be easily relaxed.

All algorithms for geometric graphs that we discuss in this survey are randomized. Even though the algorithms can be derandomized, and the final results are stated in deterministic versions as well, the randomized versions of the algorithms are more natural to present and are simpler.

For a given undirected graph G, a *traveling salesman tour* (TST) is any Hamiltonian cycle in G; a *traveling salesman path* is any Hamiltonian path in G. For a given set S of points, a TST is any traveling salesman tour for the complete graph on S. For a given graph G with cost on the edges, or for a set of points S in a metric space, the *traveling salesman problem* (TSP) is to find a TST in G or for S, respectively, that has a minimum total cost of the edges. For simplicity of our presentation, we define a TST for a set of *two* points to be the edge connecting these points.

We use term L^d*-cube* with $L \in \mathbb{R}$ to denote any axis-parallel d-dimensional cube in $\mathbb{R}^d$ of side-length L in all d dimensions. A *bounding box* of the input multiset of points in $\mathbb{R}^d$ is any L^d-cube in $\mathbb{R}^d$ enclosing these points.

The perturbation. In our algorithms for multiconnectivity problems, we first *perturb* the input instance so that each node lies on the unit grid and every inter-node distance is at least 8. We begin with rescaling the input so that the smallest bounding box L^d has $L = \mathcal{O}(n^3)$ being a power of two. Next, we move every point to the nearest point on the unit grid whose all coordinates are multipliers of 8 (which may merge some points). Then, it is easy to see that the perturbation ensures that any k-VCSS for the original input instance is now mapped into a k-VCSS whose cost (after rescaling) differs by at most an ε fraction. Therefore, if we can find a $(1+\mathcal{O}(\varepsilon))$-approximation for the k-VCSS problem for the perturbed instance, then we can directly obtain a $(1+\mathcal{O}(\varepsilon))$-approximation for the k-VCSS problem for the original instance. Because of that, from now on we assume that all input points have integer coordinates, lie in the cube $[0, L]^d$ with $L = \mathcal{O}(n^3)$ being a power of two, and the distance between any two points is either 0 or is at least 8. (We chose $L = \mathcal{O}(n^3)$ for convenience only; a much smaller L would be enough.)

The dissection. The concept of space partitioning via *dissections* (quadtrees) and *shifted dissections* plays the key role in all our algorithms. Following [13], we define the geometric partitioning of a bounding box as follows. A (2^d*-ary*) *dissection* of the bounding box L^d of a set of points in $\mathbb{R}^d$ is its recursive partitioning into smaller sub-cubes, called *regions*. Each *region* U^d of volume larger than 1 is recursively partitioned into 2^d regions $(U/2)^d$. A 2^d*-tree* with respect to the (2^d-ary) dissection is a tree whose root corresponds to the bounding box, and whose other nonleaf nodes correspond to the regions containing at least two points from the input multiset. For a nonleaf node v of the tree, the nodes corresponding to the 2^d regions partitioning the region corresponding to v are the children of v in the tree. Note that the dissection has $\Theta(L^d)$ regions and its recursion depth is logarithmic in L. Furthermore, if L is a power of 2, the boundaries of all regions in the dissection have integer coordinates, and thus they are in the unit grid.

For any d-vector $\mathbf{a} = (a_1, \ldots, a_d)$, where all a_i are $0 \leq a_i \leq L$, the $\mathbf{a}$*-shifted dissection* [13] of a set X of points in the bounding box $[0, L]^d$ in $\mathbb{R}^d$ is the result of shifting all the regions in the dissection of X in the bounding box $[0,2L]^d$ by the vector $(-\mathbf{a})$ (see Figure 51.1). The $\mathbf{a}$*-shifted* 2^d*-ary tree* with respect to the $\mathbf{a}$-shifted dissection is defined analogously. A *random shifted dissection* of a set of points X in a cube L^d in $\mathbb{R}^d$ is an $\mathbf{a}$-shifted dissection of X with $\mathbf{a} = (a_1, \ldots, a_d)$ and the elements $a_1, \ldots, a_d$ chosen independently and uniformly at random from $\{0, 1, \ldots, L\}$.

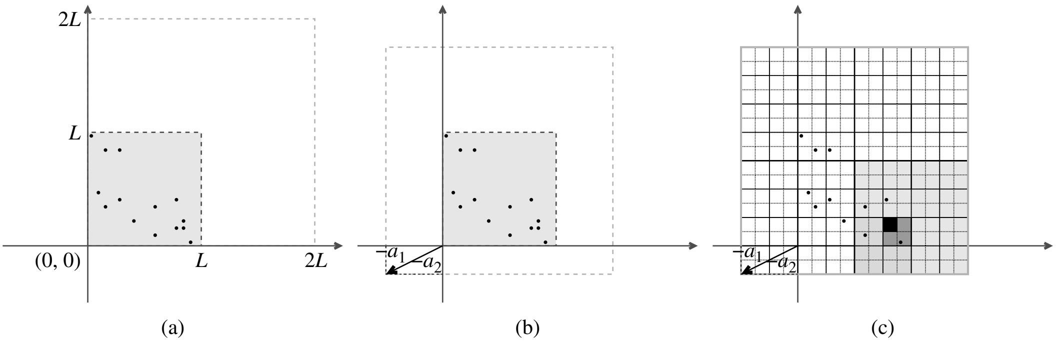


FIGURE 51.1 Shifted dissection of a set of points in the bounding box $[0, L]^d$ in $\mathbb{R}^2$ (with $\mathbf{a} = (a_1, a_2)$).

51.3 First Approach: Polynomial-Time "Pseudo-Approximation" Schemes

After the development of polynomial-time approximation schemes for the TSP problem due to Arora [13] and Mitchell [21], it seemed to be almost a straightforward task to extend their schemes to obtain a PTAS for multiconnectivity problems, or at least for the most basic 2-VCSS and 2-ECSS problems. However, it turned out that the schemes, which work very well for TSP and for some number of related problems, including Minimum Steiner Tree, Min-Cost Perfect Matching, **k**-TSP, and k-MST, could not be extended in a simple way. The reason was that in all these approximation schemes, the key step was to find a low-cost solution which uses Steiner points. While (by the triangle inequality) a Steiner point can be removed from a TST without any increase of its cost, such a transformation is impossible for k-VCSS and k-ECSS problems, e.g., a minimum-cost 2-VCSS for a point set S in $\mathbb{R}^2$ can have cost as much as $\frac{\sqrt{3}}{2}$ times larger than a Steiner 2-VCSS for S [30].

Despite this difficulty, Czumaj and Lingas [23] showed that one can apply the approach developed by Arora [13] to design a "pseudo-approximation scheme": an algorithm that finds a Steiner k-VCSS for a point set S whose cost is at most $(1 + \varepsilon)$ times larger than the cost of a minimum-cost S-VCSS for S. In other words, the algorithm finds a solution with Steiner points that has cost not much larger than an optimal solution that uses no Steiner points. Even though this is only a pseudo-approximation scheme and not a PTAS, in this section we shall present this algorithm in detail, because the underlying ideas of this approach are used later in all other algorithms we discuss in this survey.

On a very high level, the approach of Arora [13] (see also Refs. [21,31,32]) is a clever combination of the divide-and-conquer method with the dynamic programming approach, and as such, it follows a design of many classical PTASs. For the multiconnectivity problems, similarly as Arora, we hierarchically partition the cube containing the input points (via random shifted dissection) into regions, and then prove the key technical result, that there is an approximate solution to the problem that can cross the boundaries of each region only in prespecified points a bounded number of times (*Structure Theorem*). The Structure Theorem states that for any problem instance there is a $(1 + \mathcal{O}(\varepsilon))$-approximation that satisfies some basic local property: it is *m-portal respecting* and *r-light* (see the definitions below). The Structure Theorem is proven by taking an optimal solution to the problem and applying a sequence of transformations that increases the cost of the resulting graph and at the same time makes it m-portal respecting and r-light. Once the Structure Theorem is proved, a dynamic programming procedure finds in a polynomial-time an almost optimal solution that satisfies the basic local property. The dynamic programming procedure combines optimal partial solutions within regions into an optimal global solution under the crossing restrictions. To combine solutions efficiently, we derive a k-connectivity characteristic of a spanning subgraph within a region solely in terms of the set of prespecified points on the region boundary included in its vertex set. In our crucial theorem, we show that the connectivity characteristic of a union of two adjacent subgraphs

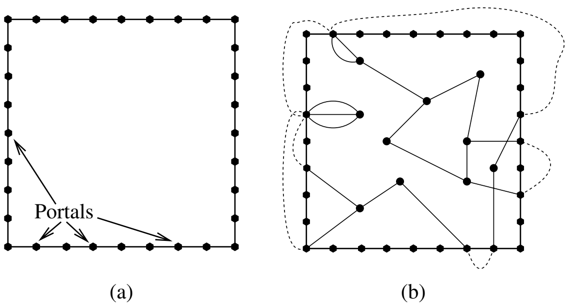


FIGURE 51.2 Portals and a portal-respecting graph.

can be computed from the connectivity characteristics of the subgraphs. This allows us to set up a dynamic programming procedure computing a $(1 + \mathcal{O}(\varepsilon))$-approximation of minimum-cost Euclidean graph, which is k-vertex or k-edge connected and obeys the crossing restrictions.

Below we discuss this approach in more detail; we focus only on the k-VCSS problem and note that the extension to the k-ECSS problem is straightforward.

51.3.1 Special Forms of Geometric Graphs

51.3.1.1 *m*-Portal-Respecting Graphs

For every integer m, an *m-regular* set of *portals* in a $(d-1)$-dimensional region facet U^{d-1} is an orthogonal lattice of m points in the facet where the spacing between the portals is $(U+1) \cdot m^{-1/(d-1)}$. A graph is *m-portal-respecting* (with respect to a shifted dissection) if whenever it crosses a facet in the dissection, it does so at a portal. Observe that this restriction forces us to assume that an m-portal-respecting graph may have to *bend* some of its edges, that is, an edge may deviate from being a straight line connecting its endpoints and be rather a *straight-line path* between the endpoints. If we are allowed to bend the ends, then for any graph in a dissection it is easy to make it m-portal-respecting by moving every crossing of every facet to its nearest portal (see Figure 51.2). Arora [13] proved the following result that transforms a graph into an m-portal-respecting one at a small cost increase and without changing the connectivity.

Lemma 51.1

Let G be a geometric graph in $\mathbb{R}^d$ for a set of (perturbed) points contained in a bounding box L^d. Pick a random shifted dissection of L^d. Then, one can transform G into an m-portal-respecting graph by moving each crossing of each facet to its nearest portal so that the expected increase of the cost of the resulting graph is at most $\mathcal{O}(d \log L m^{-1/(d-1)}) \cdot \mathrm{cost}(G)$.

Proof

Pick any edge (v, u). By the definition of the dissection, edge (v, u) crosses the facets in the dissection at most $\mathcal{O}(\sqrt{d}) \cdot c(v, u)$ times, where $c(v, u)$ is the cost of the edge (v, u).[1] To make this edge m-portal-respecting, we move each crossing of a facet to the nearest portal, which involves bending the edge that might increase its length. If the facet has side-length $L/2^i$, then this increases the distance by at most $\mathcal{O}(\sqrt{d}\, L/2^i)\, m^{-1/(d-1)}$, since the interportal distance is $\mathcal{O}(L/2^i)\, m^{-1/(d-1)}$. Because we have chosen the dissection at random, the probability that a given facet has side-length $L/2^i$ is $\mathcal{O}(2^i/L)$. Hence, the expected increase of the cost of a given edge (v, u) is

$$\sum_{i=0}^{\log L} \mathcal{O}\big(\sqrt{d}\, 2^i/L\big) \cdot \mathcal{O}(L/2^i)\, m^{-1/(d-1)} = \mathcal{O}\big(\sqrt{d} \log L \cdot m^{-1/(d-1)}\big)$$

[1] The number of crossings of the facets in the dissection is upper bounded by a constant times the ℓ_1 distance between v and u, and this is upper bounded by $\mathcal{O}(\sqrt{d})$ times the ℓ_2 distance between v and u, that is, $\mathcal{O}(\sqrt{d}) \cdot c(v, u)$.

The same arguments can be applied to all of at most $\mathcal{O}(\sqrt{d}) \cdot c(v, u)$ dissection crossings by any edge (v, u). Therefore, the expected increase of the cost of the entire graph is at most

$$\sum_{(v,u)} \left(\mathcal{O}\left(\sqrt{d}\right) \cdot c(v, u)\right) \cdot \mathcal{O}\left(\sqrt{d} \log L \cdot m^{-1/(d-1)}\right) = \mathcal{O}\left(d \log L \cdot m^{-1/(d-1)}\right) \cdot \mathrm{cost}(G) \qquad \square$$

Note that in our applications we require this error term to be at most an $\mathcal{O}(\varepsilon)$ factor of the cost of the optimal solution, and therefore we set $m = (\mathcal{O}(d \log L/\varepsilon))^{d-1}$. Using the transformation from Lemma 51.1, from now on, we assume that we consider a geometric graph that is m-portal-respecting with $m = (\mathcal{O}(d \log L/\varepsilon))^{d-1}$.

Special forms of geometric graphs: r-light graphs. We say a geometric graph is *r-light* (with respect to a shifted dissection) if for each region in the dissection there are at most r edges *crossing any of its facets.*

51.3.2 Dynamic Programming and Finding an Optimal *m*-Portal-Respecting *r*-Light Solution

In our presentation, we begin from the end and discuss first the goal of our analysis. In the following section, we show that for any set S of n (perturbed) points in $\mathbb{R}^d$ that are contained in a bounding box L^d, if we choose a random shifted dissection of L^d, then with a good probability there is an m-portal-respecting r-light (for the dissection chosen) Steiner k-VCSS for S whose cost is at most $(1 + \mathcal{O}(\varepsilon))$ times the cost of the optimal k-VCSS for S, for appropriated values of m and r. How can we use this existential result? The key observation is that if we restrict ourself to m-portal-respecting r-light graphs then we can use dynamic programming to actually find an almost optimal Steiner k-VCSS efficiently! In what follows, we briefly discuss main ideas of this result (see Ref. [23] for more details).

The key idea is that the subproblem (finding an optimal Steiner k-VCSS) inside a region in the dissection can be solved independently of the subproblems in other regions provided that we know which portals are used by the edges of the graph and the structure of the external k-connectivity properties outside that region. External k-connectivity properties outside a region are defined in terms of the portals used: if a portal is used by the graph outside the current region, we want to know which connections with other portals it supports. The concept of *connectivity characteristic* developed in Ref. [23] aims at maintaining this structural properties of graphs contained in any region.

Let us define an *internal interface* of a region Q to be any multiset of at most m portals, such that for every facet of Q, the total multiplicity of all portals (in the multiset) is upper bounded by r. Since our goal is to find an m-portal-respecting and r-light Steiner k-VCSS with low cost, we do not know its structure in advance (except that it is m-portal-respecting and r-light) and hence in our algorithm we have to consider all possible internal interfaces. Note that for m-portal-respecting and r-light graphs the number of internal interfaces of a region is at most $m^{\mathcal{O}(d\,r)}$.

Next, for any region Q and any given internal interface of Q, we define a *connectivity characteristic* to be a description of routing properties within the region and requirements on the routing properties in the complementary graph from the point of view of portals needed to preserve k-connectivity. Let P be the multiset of points in the portals used in the internal interface. The connectivity characteristic consists of three parts corresponding to different aspects of k-vertex connectivity requirements for the graph:

- Requirements for *internal connectivity*: what configurations of external disjoint paths ought to be outside the region to make any pair of vertices within region k-vertex connected; since for any pair of points in Q, all sets of disjoint paths leaving Q must traverse through the portals, each such set of vertex-disjoint paths can be encoded by a matching in the complete graph on P.
- Requirements for *internal/external connectivity*: what configurations of disjoint paths ought to be inside and outside Q to ensure that any vertex inside region Q has k vertex-disjoint paths to any vertex outside the region; this can be encoded by a set of pairs consisting of a matching in the

complete graph on P and a subset of portals (that is used to encode the parts of the paths from a vertex inside Q to the first portals, before they leave Q).

- Requirements for *external connectivity*: what configurations of internal disjoint paths ought to be inside Q to ensure that any pair of vertices outside Q are connected by k-vertex-disjoint paths; this can be encoded by matchings in the complete graph on P.

One can show that for a given region and its internal interface, there are at most $2^{(d\,r)^{\mathcal{O}(d\,r)}}$ connectivity characteristics.

The goal of the dynamic programming procedure is to determine for each region Q, for each possible internal interface of Q, and for each possible connectivity characteristic of Q, an (almost) optimal (m, r)-light graph within the region using given internal interface and having given connectivity characteristic. We maintain a lookup table that, for each region, each internal interface, and each connectivity characteristic, stores the optimal way to solve the subproblem inside the region. The lookup table is created bottom-up and the efficiency of this procedure relies on the efficiency of computing the connectivity characteristic for a region from its 2^d subregions one level down in the 2^d-dissection tree. One can find a minimum-cost graph within region Q having a given characteristic by combining minimum-cost graphs within subregions of Q, and this can be done in time $m^{d\,2^d\,r} \cdot 2^{(d\,r)^{\mathcal{O}(d\,r)}}$. This approach has to be refined for regions corresponding to the leaves in the 2^d-dissection tree, where we have to find an optimal graph directly. Unfortunately, since we do not know the locations of Steiner points in an optimal solution, we can only find an approximate solution within every leaf region. Still, this is enough to conclude with the following result (see Ref. [23] for more details):

Lemma 51.2

Let S be a (perturbed) point set in $\mathbb{R}^d$ contained in a bounding box L^d and with minimum nonzero inter-distance at least 8. *Let m and r be integer parameters. Then, in time $n \cdot \log L \cdot m^{\mathcal{O}(d\,2^d\,r)} \cdot 2^{(d\,r)^{\mathcal{O}(d\,r)}}$ one can find a $(1 + \mathcal{O}(\varepsilon))$-approximation of a minimum-cost m-portal-respecting r-light Steiner k-VCSS for S.* □

51.3.3 Patching Lemma: Reducing Number of Crossings Using Steiner Points

In this section, we discuss a *patching* procedure (initially used by Arora [13] for TSP), which is a key ingredient of our result that for any set of points and for a random shifted dissection of its bounding box, there is always an m-portal-respecting r-light Steiner k-VCSS for S whose cost is low. The patching procedure takes any facet crossed by more than k edges and patches the crossings to reduce the number of crossings to at most k, by augmenting the original graph with new Steiner vertices and new edges (line segments).

Lemma 51.3 (Patching Lemma)

Let $\mathcal{F}$ be a $(d-1)$-dimensional facet of side length W and let H be any Steiner k-VCSS (for some point set S) that crosses $\mathcal{F}$ exactly ℓ times, $\ell > k$. Then, one can break edges of H in all but k of the crossings and add to H new Steiner vertices (that lie infinitesimally close to $\mathcal{F}$) and line segments of total cost at most $\mathcal{O}(kW\ell^{1-1/(d-1)})$ such that H changes into a k-VCSS H^* for S that crosses $\mathcal{F}$ at most k times.

Proof

Let $x_1, \ldots, x_\ell$ be the points at which H crosses the $(d-1)$-dimensional facet W^{d-1}-cube $\mathcal{F}$. For each i, $1 \le i \le \ell$, break the edge (y_i, z_i) crossing $\mathcal{F}$ at x_i into two parts, one on each side of $\mathcal{F}$; we assume that all vertices $y_1, \ldots, y_\ell$ are on the same side of $\mathcal{F}$. We consider $2k + 4$ copies of each x_i, denoted by $x_{i,j}^{+}$, and $x_{i,j}^{-}$ with $0 \le j \le k+1$; $k+2$ copies for each side of $\mathcal{F}$. We assume that all copies are at distance zero from each other.

Now, we define H^*. H^* is obtained from H by removing all the edges crossing $\mathcal{F}$, and inserting the vertices $\{y_1, \ldots, y_\ell\} \cup \{z_1, \ldots, z_\ell\} \cup \bigcup_{1 \le i \le \ell \;\&\; 0 \le j \le k+1} \{x_{i,j}^{+}\} \cup \bigcup_{1 \le i \le \ell \;\&\; 0 \le j \le k+1} \{x_{i,j}^{-}\}$ and eight groups

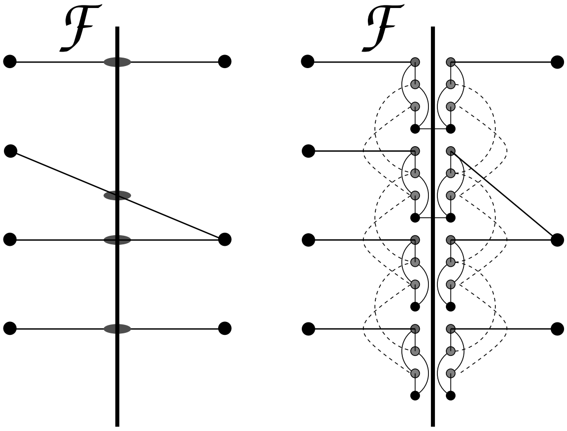


FIGURE 51.3 Graph H^* constructed in the Patching Lemma. Dotted lines corresponds to the traveling salesman paths. In this example $d = 2$, $\ell = 4$, and $k = 2$.

of edges (see Figure 51.3):

(i) two halves of each edge crossing $\mathcal{F}$ in H in the form of the edges $\{y_i, x^+_{i,0}\}$ and $\{x^-_{i,0}, z_i\}$, for all $1 \le i \le \ell$,
(ii) edges crossing $\mathcal{F}$ that connect $x^+_{i,k+1}$ with $x^-_{i,k+1}$, for all $1 \le i \le k$,
(iii) k edges connecting $x^+_{i,0}$with $x^+_{i,j}$, for all $1 \le i \le \ell$, $1 \le j \le k$,
(iv) edges connecting $x^+_{i,k+1}$ with $x^+_{i,j}$, for all $1 \le i \le \ell$, $1 \le j \le k$,
(v) edges connecting $x^-_{i,0}$ with $x^-_{i,j}$, for all $1 \le i \le \ell$, $1 \le j \le k$,
(vi) edges connecting $x^-_{i,k+1}$ with $x^-_{i,j}$, for all $1 \le i \le \ell$, $1 \le j \le k$,
(vii) edges of a traveling salesman path for $\bigcup_{1 \le i \le \ell}\{x^+_{i,j}\}$, for all $1 \le j \le k$, and
(viii) edges of a traveling salesman path for $\bigcup_{1 \le i \le \ell}\{x^-_{i,j}\}$, for all $1 \le j \le k$.

(Observe that all edges in groups (ii)–(vi) have cost zero (infinitesimally small), because we assumed that for every i and j, $1 \le i < \ell$, $0 \le j \le k+1$, all nodes $x^+_{i,j}$ and $x^-_{i,j}$ are at distance zero from each other.)

It is easy to see that the cost of the nonzero length edges in $H^* \setminus H$ is bounded from above by the cost of the edges in H plus the cost of $2\,k$ traveling salesman paths for the point sets $\bigcup_{1 \le i \le \ell}\{x^+_{i,j}\}$, $\bigcup_{1 \le i \le \ell}\{x^-_{i,j}\}$, $j = 1, \ldots, k$, respectively. Now, a well-known result about geometric TSP (see, e.g., Chapter 6 in Ref. [33]) implies that for any set of ℓ points contained in a $(d-1)$-dimensional W^{d-1} cube, there is a traveling salesman path of total length smaller than $\mathcal{O}(W\ell^{1-\frac{1}{d-1}})$. Therefore, we can conclude that the total additional cost is bounded by $\mathcal{O}(k\,W\ell^{1-\frac{1}{d-1}})$.

Finally, it is not hard to show that H^* satisfies the vertex-connectivity requirements. □

51.3.4 Structure Theorem: There Is Always a Good *r*-Light Steiner *k*-VCSS

Now, we are ready to present the first Structure Theorem for the k-VCSS problem. This theorem compares the cost of an m-portal-respecting r-light Steiner k-VCSS for a set of points with the cost of an optimal k-VCSS for this set of points, where the optimal solution is not allowed to use Steiner points.

Theorem 51.1 (Structure Theorem)

Let S be a (perturbed) point set in $\mathbb{R}^d$ contained in a bounding box L^d and with minimum nonzero interdistance at least 8*. Pick a random shifted dissection of L^d. Then with probability at least* 0.9*, there is an m-portal-respecting r-light Steiner k-VCSS for S whose cost is at most $(1 + \mathcal{O}(\varepsilon))$-times the optimal k-VCSS for S, where $m = (\mathcal{O}(d \log L/\varepsilon))^{d-1}$ and $r = (\mathcal{O}(\sqrt{d}\,k/\varepsilon))^{d-1}$.*

The proof of the Structure Theorem follows from the Patching Lemma above by repeatedly patching the original graph in an appropriated order of facets, following the original approach of Arora. This part of the analysis is technical and subtle, and we only sketch it here; a reader interested in more detail is refered to Ref. [23] or Refs. [13,31].

Sketch of the proof

The idea is to transform an optimal k-VCSS for S into an r-light Steiner k-VCSS for S of low cost by applying the Patching Lemma 51.3 to every facet which is crossed too often. Lemma 51.3 ensures that the resulting graph is an r-light Steiner k-VCSS for S. However, since its every application increases the cost of the resulting graph, it is crucial to show that the expected cost of the resulting graph is at most $(1 + \mathcal{O}(\varepsilon))$-time the optimal k-VCSS for S. If we prove this claim, then the lemma follows by applying Lemma 51.1 and by Markov inequality.

We bound the total cost of the new edges resulting from invoking the Patching Lemma by charging their cost to grid hyperplanes. For every facet in the dissection we charge the cost of removing the excess of the edges crossing the facet to the grid hyperplane that contains the facet. We show that the expected cost charged to a grid hyperplane $\mathcal{H}$ is at most $\varepsilon\, t(\mathcal{H})/(2\sqrt{d})$, where $t(\mathcal{H})$ is the number of crossings of the hyperplane $\mathcal{H}$ by the optimal k-VCSS for S. Now, the result follows by the linearity of expectations and by the fact that $\sum_{\mathcal{H}} t(\mathcal{H})$ is at most $2\sqrt{d}$ times the cost of the optimal k-VCSS for S (this result is obtained by a well-known relation between the ℓ_1 and ℓ_2 norms).

Let us fix a grid hyperplane $\mathcal{H}$ perpendicular to some coordinate axis. Note that within the bounding box L^d, $\mathcal{H}$ forms a L^{d-1} cube. We apply the Patching Lemma to all facets of the dissection that belong to $\mathcal{H}$. We first begin with the smallest facets, and then consider the facets in the increasing order of their sizes. Let c_j be the number of facets in $\mathcal{H}$ of side length $L/2^j$ for which patching has been invoked. For $\ell \le c_j$, let $t_{j,\ell}$ be the number of crossings of the ℓth facet of side length $L/2^j$ for which patching has been applied. Observe that for the ℓth facet of side length $L/2^j$ for which patching has been applied, the total cost of the new edges added by the Patching Lemma is upper bounded by $\mathcal{O}(k\,(L/2^j)\,(t_{j,\ell})^{1-1/(d-1)})$. Therefore, if the largest facet in the hyperplane $\mathcal{H}$ has side-length $L/2^i$, then the total cost of the new edges added by applying the Patching Lemma to all facets in $\mathcal{H}$ is upper bounded by:

$$\mathcal{O}\left(\sum_{j=i}^{\log L}\sum_{\ell=1}^{c_j} k\,\left(L/2^j\right)\,(t_{j,\ell})^{1-1/(d-1)}\right) \tag{51.1}$$

Next, we study the expected cost as above, where the expectation is taken over shifts chosen in the random shifted dissection. Let us assume that the grid hyperplane $\mathcal{H}$ is perpendicular to the sth coordinate axis. Let us fix the random vector $\mathbf{a} = (a_1, \ldots, a_d)$ used to determine the random shifted dissection in which all elements are fixed with the exception of a_s, which is kept random. We observe that the random shift in the dissection depends only on the value of a_s, and therefore, if $a_1, \ldots, a_{s-1}, a_{s+1}, \ldots, a_d$ are fixed, then the probability that the largest facet in the hyperplane $\mathcal{H}$ has side-length $L/2^i$ is $\mathcal{O}(2^i/L)$. Furthermore, one can show that the values of c_j and $t_{j,\ell}$ are independent of a_s. Therefore, the expected cost of all edges added by applying patching to all facets in $\mathcal{H}$ is at most:

$$\begin{aligned}\sum_{i=0}^{\log L}\mathcal{O}(2^i/L)\cdot\mathcal{O}\left(\sum_{j=i}^{\log L}\sum_{\ell=1}^{c_j} k\,(L/2^j)\,(t_{j,\ell})^{1-1/(d-1)}\right) &\le \mathcal{O}\left(\sum_{j=0}^{\log L}\sum_{\ell=1}^{c_j} k/2^j\,(t_{j,\ell})^{1-1/(d-1)}\sum_{i=0}^{j}2^i\right)\\ &= \mathcal{O}(k)\cdot\sum_{j=0}^{\log L}\sum_{\ell=1}^{c_j}(t_{j,\ell})^{1-1/(d-1)}\end{aligned}$$

Since $t_{j,\ell} \ge r+1$, the bound above is maximized when each $t_{j,\ell} = r+1$, and therefore it is bounded by $\mathcal{O}(k)\cdot(r+1)^{1-1/(d-1)}\cdot\sum_{j=0}^{\log L} c_j$. Now, we need a good upper bound for $\sum_{j=0}^{\log L} c_j$. Since each application of the Patching Lemma reduces the number of crossings of $\mathcal{H}$ by at least $r+1-k$, the definition of $t(\mathcal{H})$ yields:

$$\sum_{j=0}^{\log L} c_j \le \frac{t(\mathcal{H})}{r+1-k}$$

Therefore, the expected cost of all edges added by applying patching to all facets in $\mathcal{H}$ is upper bounded by:

$$\mathcal{O}(k) \cdot (r+1)^{1-1/(d-1)} \cdot \sum_{j=0}^{\log L} c_j \le \mathcal{O}\left(\frac{k\,(r+1)^{1-1/(d-1)}\,t(\mathcal{H})}{r+1-k}\right)$$

We set $r = (\mathcal{O}(\sqrt{d}\,k/\varepsilon))^{d-1}$ to upperbound this by $\varepsilon\, t(\mathcal{H})/(2\sqrt{d})$. By our arguments above, this implies that the expected cost of all edges added by applying the Patching Lemma (which results in a transformation of the graph into an r-light one) to an optimal k-VCSS for S is at most ε times the cost of the optimal k-VCSS for S.

Finally, we have to transform the graph into m-portal-respecting. We apply the construction presented in Lemma 51.1 with the value of $m = (\mathcal{O}(d\,\log L/\varepsilon))^{d-1}$. Since, by Lemma 51.1, this construction increases in expectation the cost of the graph by at most a factor of $\mathcal{O}(\varepsilon)$, the final result follows. □

51.3.5 Final Result: "Pseudo-Approximation" Schemes for Multiconnectivity Problems

The results from the previous sections (Lemma 51.2 and Theorem 51.1) are summarized in the following theorem.

Theorem 51.2

Let k and d be any integers, k, d ≥ 2, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm which finds a Steiner k-VCSS for S, whose cost is at most $(1+\varepsilon)$-time the optimal k-VCSS for S, in time $n(\log n)^{(\mathcal{O}(\sqrt{d}\,k/\varepsilon))^{d-1}} \cdot 2^{(d\,k/\varepsilon)^{(\mathcal{O}(\sqrt{d}\,k/\varepsilon))^{d-1}}}$ with probability at least 0.9 .

Furthermore, within the same running time one can find a Steiner k-ECSS for S whose cost is at most $(1+\varepsilon)$-time the optimal k-ECSS for S. Also, all these algorithms can be derandomized in polynomial time.

Observe that when all d, k, and ε are constant, the running time of the randomized algorithm is $n \cdot (\log n)^{\mathcal{O}(1)}$. When d is a constant and k and ε are arbitrary, then the running time is $n \cdot (\log n)^{(k/\varepsilon)^{\mathcal{O}(1)}} \cdot 2^{2^{(k/\varepsilon)^{\mathcal{O}(1)}}}$.

51.4 PTAS for Geometric Multiconnectivity Problems

The results from the previous section are certainly not fully satisfactory, and a natural question arises if we can obtain a similar result *without using Steiner points* in the solution. In this section, we discuss in detail how one can modify the approach from Section 51.3 to obtain a PTAS for geometric multiconnectivity problems. Even if this method can be seen as a generalization of the approach developed initially by Arora [13], the details of the new construction are significantly different than those used for TSP and related problems. The material in this section is based on Ref. [28], an updated and improved version of Refs. [18,24].

The main idea of the PTAS is similar to that from the previous section: we want to prove a result of the form similar to that from Structure Theorem (Theorem 51.1). However, this time we want to make sure that no new Steiner points difficult to remove are created. We achieve this goal by aiming at a variant of the Structure Theorem that does not require the resulting graph to be r-light but only *r-locally-light*, see the definition below. The difference between these two requirements is insignificant for the dynamic programming phase, but it is critical in our analysis: as we show in our main theorem, there is always an almost optimal k-VCSS for a set of points in $\mathbb{R}^d$ that is m-portal-respecting and r-locally-light for small values of m and r. Before we proceed on, we begin with introducing some new notation.

Relevant crossings and vital edges. A crossing of an edge with a region facet of side length W in a dissection is called *relevant* if it has exactly one endpoint in the region and its length is at most $2\sqrt{d}\,W$. For a given region Q in a shifted dissection, any edge having exactly one endpoint in Q is called *vital* (for Q).

Special forms of geometric graphs: r-gray and r-locally-light graphs. We say a geometric graph is *r-gray* (with respect to a shifted dissection) if for each region in the dissection there are at most r *relevant crossings.* A graph is *r-locally-light* (with respect to a shifted dissection) if each region in the dissection has at most r vital edges.

Augmented traveling salesman tours. A kth *power* of a graph G is obtained by augmenting G by the edges whose endpoints are connected by paths consisting of at most k edges in G. For any set S and ℓ, an *ℓ-augmented traveling salesman tour on S* is either a clique on S if $|S| \leq 2\,\ell$, or the ℓth power of some TST on S if $|S| \geq 2\,\ell + 1$.

51.4.1 Transformation Lemmata

In this section, we present a variant of the Structure Theorem designed to deal with the problem of finding a minimum-cost k-VCSS for a set of points in $\mathbb{R}^d$. Our goal is to obtain a similar claim as the Structure Theorem (Theorem 51.1) but without the assumption that the promised graph has Steiner points. We prove this new Structure Theorem in three steps. We take an optimal solution for the minimum-cost k-VCSS problem and we modify it to a suitable form to obtain a graph that is still k-VCSS and whose cost is just slightly larger than that of the minimum-cost. In the first two steps we remove some number of edges (and thus, we do not increase the cost of the graph) to ensure that the resulting graph is first r-gray and then r-locally-light. In the third step we add replacement of the removed edges to ensure that the obtained graph is k-VCSS. The first and the third steps are randomized and they show that in expectation the cost of the resulting graph is at most $(1 + \mathcal{O}(\varepsilon))$ times the minimum-cost k-VCSS.

51.4.1.1 Local Decomposition Lemma

In this section we discuss our first key result in the analysis, the so-called Local Decomposition Lemma. The Local Decomposition Lemma aims at reducing the number of *relevant* crossings of any given facet to at most k. This procedure is very similar in the spirit to the Patching Lemma 51.3. However, unlike the previously known approaches, the Local Decomposition Lemma *does not use any Steiner points.* The key feature of this construction is that it *only removes edges* and the decision that new edges should be inserted to ensure the connectivity requirements is *delayed.* Instead, a description of properties the new edges must satisfy is provided and these edges are inserted only at the very end of the algorithm (using the TST Covering Lemma 51.7).[2]

To streamline maintaining the connectivity properties of the missing edges, we always describe missing edges in a form of k-augmenting TSTs. The idea is that in order to ensure that a set of points is k-connected it is enough to maintain its TST and then observe that the k-augmenting TST is k-connected. Furthermore, by controlling the cost of a minimum-cost TST for that set of points, we can also control an upper bound for a minimum-cost k-augmenting TST for these points. This will be important in our analysis.

[2] One can ask why do we delay inserting the new edges, e.g., in a similar situation in Arora's PTAS for TSP [13], the new edges are inserted at once, as we also do in the analysis of the Structure Theorem (Theorem 51.1). Note however that Arora [13] and others were always able to place the new edges on the facet for which the Patching Lemma is applied, which facilitates dealing with the new crossings. In the case discussed here, we do not want to create Steiner points and therefore we need to add new edges in arbitrary locations.

Lemma 51.4 **(Local Decomposition Lemma)**

Let G be an Euclidean graph on a multiset S of points in $\mathbb{R}^d$. Let $\mathcal{F}$ be a $(d-1)$-dimensional facet of side length W in a dissection of the bounding box of S. If the edges of G form ℓ relevant crossings of $\mathcal{F}$, then there exist a subgraph G^* of G, and two disjoint subsets S_1 and S_2 of S, such that

- *there are at most $2\,k^2$ relevant crossings of $\mathcal{F}$ in G^*,*
- *there are a* TST *on S_1 and a* TST *on S_2 such that the cost of each is upper bounded by $\mathcal{O}(d\,W\,\ell^{1-\frac{1}{d}})$, and*
- *if G is a k-VCSS on S, then the graph H^* resulting from the graph G^* by adding* ***any*** *k-augmented* TST *on S_1 and* ***any*** *k-augmented* TST *on S_2, is a k-VCSS on S.*

Remark 51.3

There are three key differences between the Local Decomposition Lemma and the Patching Lemma 51.3: (i) the Local Decomposition Lemma does not introduce any new points to the obtained graph, (ii) it reduces only the number of relevant crossings, leaving the number of arbitrary crossings possibly arbitrarily large, and (iii) it does not produce a k-VCSS on S, but rather it says that one can build one by adding some additional edges.

Remark 51.4

For a given TST *$\mathcal{T}$ on X it is easy to construct a k-augmented* TST *$\mathfrak{T}^{\langle k\rangle}$ on X such that the cost of $\mathfrak{T}^{\langle k\rangle}$ is at most $\binom{k+1}{2} \le 2\,k^2$ times larger than the cost of $\mathcal{T}$ and each hyperplane $\mathcal{H}$ (which does not contain any edge from $\mathcal{T}$) is crossed by the edges of $\mathfrak{T}^{\langle k\rangle}$ at most $\binom{k+1}{2} \le 2\,k^2$ times more than it is crossed by the edges of $\mathcal{H}$.*

Proof

We can assume $\ell > 2\,k^2$. We first construct the subgraph G^* and the subsets S_1 and S_2, and then briefly argue about their properties.

Let $\mathcal{E}$ be the set of the ℓ edges of G forming the ℓ *relevant* crossings with $\mathcal{F}$. We define $S_1 = \{x_1, \ldots, x_\ell\}$ as the set of endpoints of the edges in $\mathcal{E}$ in the first half-space induced by $\mathcal{F}$ and $S_2 = \{y_1, \ldots, y_\ell\}$ as the corresponding set of endpoints of these edges in the other half-space. Next we define G^*. G^* is obtained by removing from G a subset of the edges in $\mathcal{E}$. Let $\mathbb{M}$ be a maximum cardinality subset of $\mathcal{E}$ such that no two edges in the subset are incident. Let $q = \min\{k, |\mathbb{M}|\}$. Then, we define the set $\mathcal{E}^*$ of edges in $\mathcal{E}$ that will remain in G^* to consist of

- the first q edges of $\mathbb{M}$, and
- if $q < k$, then, additionally, for each endpoint v of each edge from $\mathbb{M}$ we add to $\mathcal{E}^*$ $\min\{k-1, deg_{\mathcal{E}}(v) - 1\}$ edges in $\mathcal{E}\backslash\mathbb{M}$ incident to v, where $deg_{\mathcal{E}}(v)$ is the number of edges in $\mathcal{E}$ incident to v.

Now, the graph G^* is obtained from G by removing the edges in $\mathcal{E} \setminus \mathcal{E}^*$.

To complete the proof, we must show that G^*, S_1, and S_2 satisfy the properties promised in the lemma. Clearly, $\mathcal{E}^*$ is of size at most $2\,k^2$, and hence there are at most $2\,k^2$ relevant crossings of $\mathcal{F}$ in G^*. Furthermore, each of S_1 and S_2 consists of at most ℓ vertices that are contained in a bounding box of size $\mathcal{O}(\sqrt{d}\,W)$. (Indeed, since the vertices in S_1 and S_2 are endpoints of relevant crossings $\mathcal{F}$, their distance from $\mathcal{F}$ is bounded by $2\sqrt{d}\,W$.) Thus, there is a TST on each of S_1 and S_2 of total length smaller than $\mathcal{O}(d\,W\,\ell^{1-\frac{1}{d}})$ (see, e.g., Section 6 in Ref. [33]).[3] The remaining properties can be also easily shown (see Refs. [18,28] for details). □

[3] Note that we need here the assumption that each of S_1 and S_2 is included in a bounding box of size $\mathcal{O}(\sqrt{d}\,W)$. In contrast, in the Patching Lemma 51.3, the points could be arbitrarily far away from each other and thus, for example, there could be no TSP on S_1 of length $o(n)$.

51.4.1.2 Weak (Too Weak) Version of Structure Theorem: Global Decomposition Lemma

With the Local Decomposition Lemma above, we can provide a weak version of the Structure Theorem that uses similar arguments as those used in the proof of Theorem 51.1. Since this formulation is too weak for our applications, our goal in the following sections will be to extend it to obtain a stronger result.

Lemma 51.5 (**Global Decomposition Lemma**)

Let S be a (perturbed) point set in $\mathbb{R}^d$ contained in a bounding box L^d and with minimum nonzero inter-distance at least 8*. Pick a random shifted dissection of L^d. Then, there is an r-gray graph G on S and a collection $\mathbb{S}$ of (possible intersecting) subsets of S such that:*

- *the cost of G is not larger than the minimum cost of k-VCSS for S,*
- *$r = (\mathcal{O}(k^2\, d^{3/2}/\varepsilon))^d$,*
- *there is a graph H consisting of (possible nondisjoint)* TSTs *on every set $X \in \mathbb{S}$ whose expected (over the choice of the random shifted dissection) total cost is at most $\mathcal{O}(\varepsilon/k^2)$ times the minimum cost of k-VCSS for S, and*
- *the graph resulting from G by adding* ***any*** *k-augmented* TSTs *on each $X \in \mathbb{S}$ is a k-VCSS on S.*

Proof

The proof of this result mimics the proof of the Structure Theorem (Theorem 51.1) with the exception of a few modifications that are caused by a different form of the Local Decomposition Lemma 51.4. We take a minimum-cost k-VCSS G_{opt} for S and apply a sequence of the Local Decomposition Lemma to make this graph r-gray. Since each application of the Local Decomposition Lemma only removes the edges from G_{opt}, the obtained graph G is a subgraph of G_{opt} and hence its cost is not larger than the minimum cost of k-VCSS for S. Furthermore, the resulting graph is r-gray by Lemma 51.4. This lemma ensures also that if we define $\mathbb{S}$ as the family of sets returned by all calls to the Local Decomposition Lemma, then by adding to G any k-augmented TSTs on all $X \in \mathbb{S}$ we obtain a k-VCSS on S.

What remains to prove is that for the sets $X \in \mathbb{S}$, the total expected costs of minimum-cost TSTs on the sets $X \in \mathbb{S}$ is at most $\mathcal{O}(\varepsilon/k^2)$ times the cost of G_{opt}. The proof of this fact mimics the analysis of the Structure Theorem (Theorem 51.1). We charge the cost of invoking the Local Decomposition Lemma to a facet contained in a grid hyperplane to that hyperplane. Then, a similar analysis implies that the expected cost of all minimum-cost TSTs on all sets $X \in \mathbb{S}$ resulting from applying the Local Decomposition Lemma to all facets contained in $\mathcal{H}$ is upper bounded by:

$$\mathcal{O}\left(\frac{d \cdot (r+1)^{1-1/d} \cdot t(\mathcal{H})}{r+1-2\,k^2}\right)$$

Now, if we set $r = (\mathcal{O}(k^2\, d^{3/2}/\varepsilon))^d$, then the same arguments as those used in the proof of the Structure Theorem (Theorem 51.1) imply that the expected (over the choice of the random shifted dissection) total cost of minimum-cost TSTs on all sets $X \in \mathbb{S}$ is upper bounded by $\mathcal{O}(\varepsilon/k^2)$ times the minimum cost of k-VCSS for S. □

51.4.1.3 Filtering Lemma

The Global Decomposition Lemma 51.5 transforms an arbitrary Euclidean graph G into an r-gray graph, so that certain properties of optimal k-vertex connected graphs induced by these graphs are satisfied. There are however stronger requirements for the transformed graph in order to get a PTAS. Even if after applying the Global Decomposition Lemma each facet in an r-gray graph has only $\mathcal{O}(r)$ relevant crossings, many other (longer) crossings are possible. The Filtering Lemma below transforms any r-gray graph into an r^*-locally-light one by removing a set of edges of total small cost, with the parameter r^* just slightly bigger than r.

Lemma 51.6 (**Filtering Lemma**)

Let $r \geq 1$ and let S be a (perturbed) point set in $\mathbb{R}^d$ contained in a bounding box L^d and with minimum nonzero inter-distance at least 8*. For a given shifted dissection, let $G = (S, E)$ be any r-gray graph on S.*

Then, we can find a subgraph G^* of G that is r^*-locally-light for $r^* = \mathcal{O}(r\ d\ \log(d\ k/\varepsilon))$, and such that the total cost of the edges in $G \setminus G^*$ is at most $\mathcal{O}(\varepsilon/k^2) \cdot \text{cost}(G)$.

The proof of the Filtering Lemma explores the property that if a graph is r-gray, then for every region Q of side length L in the dissection there are at most $2\ dr$ vital edges for Q whose length is in the interval $(2^j\sqrt{d}L, 2^{j+1}\sqrt{d}L)$ for every value of j. This implies that if there are many vital edges crossing any single facet then most of them (all but a small number of the heaviest edges) have small cost. Therefore, one can transform any r-gray graph into an r^*-locally-light one by deleting some number of short edges whose total cost (by careful charging arguments) is low.

51.4.1.4 TST Covering Lemma

In the previous subsections, we have transformed an Euclidean graph into the one that possesses fewer edges crossing each facet in the dissection. The key feature of the Global Decomposition Lemma and the Filtering Lemma is that after the graph transformations we are left with some (possible intersecting) sets of nodes that are to be connected in some way (either in pairs by edges or into k-augmented TSTs). The main reason of such construction was to postpone immediately connecting the nodes within each set because this could introduce many new crossings and might destroy the r-locally-lightness of the graph. The TST Covering Lemma below shows how to connect the nodes within each set without increasing the cost of the graph too significantly and without introducing too many crossings of any facet.

We need a definition of a *cover* of a superset that can be seen as a way of connecting multiple TSTs. Let $\mathbb{S}$ be a collection of (not necessarily disjoint) sets. A collection $\mathbb{S}^*$ is called a *cover* of $\mathbb{S}$ if (i) for every $X \in \mathbb{S}$ there is a $Y \in \mathbb{S}^*$ such that $X \subseteq Y$ and (ii) $\bigcup_{X \in \mathbb{S}} X = \bigcup_{Y \in \mathbb{S}^*} Y$. Now, we are ready to state the TST Covering Lemma.

Lemma 51.7 (TST Covering Lemma)

Let S be a (perturbed) point set in $\mathbb{R}^d$ contained in a bounding box L^d and with minimum nonzero inter-distance at least 8. Pick a random shifted dissection of L^d. Let $\mathbb{S}$ be a collection of (possibly nondisjoint) subsets of S. Suppose there is a graph G on S that is a union of TSTs, one for each $X \in \mathbb{S}$, of total cost cost(G). Then, there is a graph G^* such that

- *G^* is r-light with respect to the dissection, where $r = (\mathcal{O}(\sqrt{d}))^{d-1}$,*
- *there is a cover $\mathbb{S}_{G^*}$ of $\mathbb{S}$ such that G^* is the union of TSTs for each $Y \in \mathbb{S}_{G^*}$, and*
- *the expected (over the choice of the random shifted dissection) cost of G^* is at most $\mathcal{O}(\text{cost}(G))$.* □

The proof of this lemma is an extension of the PTAS for Euclidean TSP by Arora [13] and uses ideas similar to those underlined in the proof of the Structure Theorem (Theorem 51.1). (Observe that since now we need to find TSTs the appearance of Steiner points in the approach of Arora [13] does not cause any problems.)

51.4.1.5 Concluding: Structure Theorem for *k*-Vertex Connectivity

We conclude with a Structure Theorem for k-vertex connectivity that shows the existence of a low cost locally light graph. This theorem is obtained by combining Lemma 51.1, the Global Decomposition Lemma, the Filtering Lemma, and the TST Covering Lemma, when applied to a minimum-cost k-VCSS G for the input point set.

Theorem 51.5 (Structure Theorem II)

Let S be a (perturbed) point set in $\mathbb{R}^d$ contained in a bounding box L^d and with minimum nonzero inter-distance at least 8. Pick a random shifted dissection of L^d. Then with probability at least 0.9, there is an m-portal-respecting r-light k-VCSS for S whose cost is at most $1 + \mathcal{O}(\varepsilon)$ times the minimum-cost of k-VCSS for S, where $m = (\mathcal{O}(d\ \log L/\varepsilon))^{d-1}$ and $r = (\mathcal{O}(k^2\ d^{3/2}/\varepsilon))^d\ \log(k/\varepsilon)$.

51.4.2 PTAS for Euclidean *k*-Vertex and *k*-Edge Connectivity

Now, with the Structure Theorem II (Theorem 51.5) at hand, we are ready to present the "real" PTAS for the minimum-cost k-VCSS problem in geometric graphs. In Section 51.3.2, we showed how to find a $(1 + \mathcal{O}(\varepsilon))$-approximation of a minimum-cost m-portal-respecting r-light Steiner k-VCSS for a set of points (see Lemma 51.2). Similar result holds also for finding a minimum-cost m-portal-respecting r-locally-light k-VCSS for a set of points. The running time of the appropriated dynamic programming scheme is the same as that promised in Lemma 51.2, but this time we can even find an optimal solution (not an $(1 + \mathcal{O}(\varepsilon))$-approximation, as in Lemma 51.2). Therefore, we can combine this result with the Structure Theorem II (Theorem 51.5) to obtain the following result.

Theorem 51.6

Let k and d be any integers, $k, d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot (\log n)^{(k\,d/\varepsilon)^{\mathcal{O}(d)}} \cdot 2^{2^{(k\,d/\varepsilon)^{\mathcal{O}(d)}}}$ with probability at least 0.9 finds a k-VCSS for S whose cost is at most $1 + \varepsilon$ times the minimum-cost of k-VCSS for S.

Furthermore, within the same running time one can find a k-ECSS for S whose cost is at most $1 + \varepsilon$ times the minimum-cost of k-ECSS for S. Also, all these algorithms can be derandomized in polynomial time.

When the parameters ε, k, and d are constants, then the running time of the randomized algorithm is $n \cdot \log^{\mathcal{O}(1)} n$. When d and ε are constant and k is arbitrary, the running time becomes $n \cdot (\log n)^{k^{\mathcal{O}(1)}} \cdot 2^{2^{k^{\mathcal{O}(1)}}}$; when ε is arbitrary, it is $n \cdot (\log n)^{(1/\varepsilon)^{\mathcal{O}(1)}} \cdot 2^{2^{(1/\varepsilon)^{\mathcal{O}(1)}}}$. In particular, for a constant dimension d, our scheme leads to a PTAS for the minimum-cost k-VCSS and k-ECSS problems for all k such that $k \leq (\log\log n)^c$ for certain positive constant $c < 1$.

51.5 Faster PTAS for Euclidean *k*-ECSSM and 2-Connected Graphs

Czumaj and Lingas [24] showed that the approximation schemes from Section 51.4 can be improved in the special case when $k = 2$ and for the minimum-cost k-ECSSM problem. The main source of the improvement is the observation that if we knew a graph/multigraph that contains an optimal or near optimal k-VCSS (k-ECSS, k-ECSSM), then we would be able to apply similar transformations as those described in Section 51.4 to transform this graph into an r-locally-light one. Comparing to the result from the Structure Theorem II 51.5, we would gain by not having to make the graph m-portal-respecting, because dynamic programming would not have to "guess" the locations of crossings of the facets. This would potentially eliminate term m in the analysis (see Lemma 51.2), and thus greatly improve the running time.

A geometric graph G on a set of points in $\mathbb{R}^d$ is called a *t-spanner* of S, $t \geq 1$, if for any pair of points $p, q \in S$ there is a path in G from p to q of length at most t times the distance between p and q. Gudmundson et al. [34] showed that for any set S of n points in $\mathbb{R}^d$ and for any positive ε, in time $\mathcal{O}((d/\varepsilon)^{\mathcal{O}(d)} \cdot n + d \cdot n \cdot \log n)$ one can find a $(1 + \varepsilon)$-spanner of S with maximum degree $(d/\varepsilon)^{\mathcal{O}(d)}$ and with the total cost at most $(d/\varepsilon)^{\mathcal{O}(d)} \cdot \text{MST}(S)$.

For a given multigraph H, the *graph induced* by H is the graph obtained by reducing the multiplicity of each edge of H to one. The following lemma formally describes the intuition that a t-spanner contains (implicitly) a t-approximation of the minimum-cost k-ECSSM.

Lemma 51.8

Let G be a t-spanner for a point set S in $\mathbb{R}^d$ and let k be an arbitrary positive integer. Then, there exists a k-edge-connected multigraph H on S such that (i) the graph induced by H is a subgraph of G, (ii) the total cost of H is at most t times larger than the minimum-cost k-edge-connected multigraph on S, and (iii) there are no parallel edges in H of multiplicity exceeding k.

Now, with a good spanner at hand and with Lemma 51.8, we can proceed with the approach sketched before. This approach is partly inspired by the recent use of spanners to speed-up PTAS for Euclidean versions of TSP due to Rao and Smith [32]. The analysis relies on a series of transformations of a low cost and sparse $(1+\mathcal{O}(\varepsilon))$-spanner for the input point set into an r-locally-light k-edge connected *multigraph* spanning the input set and having nearly optimal cost. With some modifications of the analysis from the Structure Theorem II, one can get the following theorem.

Theorem 51.7 (Structure Theorem III)

Let S be a (perturbed) set of n points in $\mathbb{R}^d$ contained in a bounding box L^d and with minimum nonzero inter-distance at least 8*. Let G be a $(1+\varepsilon)$-spanner for S that has $n\,(d/\varepsilon)^{\mathcal{O}(d)}$ edges and has total cost $(d/\varepsilon)^{\mathcal{O}(d)} \cdot \mathrm{MST}(S)$. Choose a shifted dissection uniformly at random. Then, one can transform G into a graph G^* on S such that with probability (over the random choice of the shifted dissection) at least* 0.9

- *G^* is r-locally-light with respect to the shifted dissection, $r = kd^{\mathcal{O}(d)} + \mathcal{O}(kd^2 \log(d/\varepsilon)) + (d/\varepsilon)^{\mathcal{O}(d^2)}$, and*
- *there exists a k-edge-connected multigraph $\mathbb{H}$, which is a spanning subgraph of $\mathbb{G}$ with possible parallel edges (of multiplicity at most k) whose cost is upper bounded by $1+\mathcal{O}(\varepsilon)$ times the minimum-cost of k-ECSSM for S.*

Moreover, the transformation can be done in time $n \cdot 2^{(\mathcal{O}(\sqrt{d}))^{d-1}} + n \cdot (d/\varepsilon)^{\mathcal{O}(d)} \log n$.

Once we have the transformation defined in the Structure Theorem III (Theorem 51.7), we can use dynamic programming, similar to that described in Lemma 51.2 and in Section 51.4.2, to obtain the following lemma.

Lemma 51.9

Let S be a set of n points in $\mathbb{R}^d$ contained in a bounding box L^d and with minimum nonzero interdistance at least 8*. Consider an arbitrary shifted dissection and assume that the 2^d-ary dissection tree of S is given. Let G be an r-locally-light graph on S, where $r \geq 1$ is arbitrary. Then, a minimum-cost k-ECSSM G^* on S for which the induced graph is a subgraph of G can be found in time $n \cdot 2^{d+(kr)^{\mathcal{O}(kr)}}$.*

Therefore, if we combine the Structure Theorem III (Theorem 51.7) with Lemma 51.9, we directly obtain the following theorem.

Theorem 51.8

Let k and d be any integers, k, $d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot \log n \cdot (d/\varepsilon)^{\mathcal{O}(d)} + n \cdot 2^{2^{(k^{\mathcal{O}(1)} \cdot (d/\varepsilon)^{\mathcal{O}(d^2)})}}$, with probability at least 0.9 *finds a k-ECSSM for S whose cost is at most $1+\varepsilon$ times the minimum-cost of k-ECSSM for S. The algorithm can be derandomized in polynomial time.*

Observe that when all d, k, and ε are constant, the running time of the randomized algorithm is $\mathcal{O}(n \log n)$. When d and k are constant and ε is arbitrary, the running time becomes $n \log n\,(1/\varepsilon)^{\mathcal{O}(1)} + n\,2^{2^{(1/\varepsilon)^{\mathcal{O}(1)}}}$. When d and ε are set to be constants, then the running time is $\mathcal{O}(n \log n) + n\,2^{2^{k^{\mathcal{O}(1)}}}$.

51.5.1 2-Connected Graphs Are Not Worse than 2-Connected Multigraphs

The algorithm presented in the previous section does not work for minimum-cost k-VCSS or k-ECSS problems. The reason is that no result similar to that from Lemma 51.8 holds. However, in the special case when $k = 2$, we still can use multigraph approach to obtain a fast PTAS for the minimum-cost 2-VCSS or 2-ECSS problems. Indeed, it is known than any 2-VCSS is also a 2-ECSSM. Therefore, the minimum-cost 2-ECSSM for a set of points is not bigger than the minimum-cost 2-VCSS for the same point set. The following theorem shows that actually, we can always quickly find a 2-VCSS (and hence also 2-ECSS) that has cost not larger than that of a 2-ECSSM.

Lemma 51.10 [24,19]

A 2*-edge-connected multigraph on a set of points in* $\mathbb{R}^d$ *can be transformed in linear time into a biconnected graph on the same set of points without increasing the total cost.*

In view of this result, we could find a $(1+\varepsilon)$-approximation for the minimum-cost 2-VCSS problem by first running an algorithm for finding a $(1+\varepsilon)$-approximation of the minimum-cost 2-ECSSM and then applying Lemma 51.10. By Theorem 51.8, such randomized algorithm for the minimum-cost 2-VCSS problem runs in time $n\cdot\log n\cdot(d/\varepsilon)^{\mathcal{O}(d)}+n\cdot 2^{2^{(\mathcal{O}(d/\varepsilon))^{\mathcal{O}(d^2)}}}$. However, as Czumaj and Lingas [24,29] proved, one can obtain further speedup by improving the dynamic programming scheme from Lemma 51.9 in the special case $k=2$. For any set S of n points in $\mathbb{R}^d$ and for any Euclidean graph G on S that is r-locally-light with respect to some given shifted dissection, one can use dynamic programming to find in time $n\cdot 2^d\cdot r^{\mathcal{O}(r\,2^d)}$ a minimum-cost 2-edge-connected multigraph on S for which the induced graph is a subgraph of G. This yields the following theorem.

Theorem 51.9

Let d be any integer $d\geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm which in time $n\cdot\log n\cdot(d/\varepsilon)^{\mathcal{O}(d)}+n\cdot 2^{(d/\varepsilon)^{\mathcal{O}(d^2)}}$, with probability at least 0.9 *finds a* 2*-VCSS for S whose cost is at most* $1+\varepsilon$ *times the minimum-cost of* 2*-VCSS for S.*

The same holds for the minimum-cost 2*-ECSS problem; these algorithms can be derandomized in polynomial time.*

For constant d and arbitrary ε, the running time of the randomized algorithm is $n\,\log n\,(1/\varepsilon)^{\mathcal{O}(1)}+2^{(1/\varepsilon)^{\mathcal{O}(1)}}$.

51.6 Lower Bounds

The results discussed in previous sections show that various multiconnectivity problems have a PTAS. However, the obtained algorithms work in polynomial-time only for small values of d and k. Are these results just a sign that our methods still need to be improved or they are inherent for the multiconnectivity problems?

As for now, we still do not know if there is a PTAS for large values of k and, say, if we pick $k=\log n$ we do not know if the k-VCSS problem for geometric graphs on the plane (i.e., for $d=2$) has a PTAS or does not. However, we know that we cannot obtain a PTAS for large values of d. Our basic tool is a powerful result of Trevisan [35] that connects the inapproximability of TSP in geometric graphs with the inapproximability of TSP in the so-called 1–2 graphs. A weighted undirected complete graph G is a *1–2 graph* if each of its edges has weight either 1 or 2. It is called a *1–2-Δ graph* if it is a 1–2 graph and each of its vertices is incident to at most Δ edges of weight 1. It is easy to see that in every graph TST has cost that is not smaller than the cost of a minimum-cost 2-VCSS. The following result showing that in 1–2 graphs TST and minimum-cost 2-VCSS coincide is central for our analysis.

Lemma 51.11

In every 1–2 graph, TST is a minimum-cost 2-VCSS. □

With this result, general inapproximability results for TST in 1–2 graphs proven by Trevisan [35] directly imply similar results for the 2-VCSS problem.

Theorem 51.10 [18]

There exist constants $\Delta_0>0$ and $\varepsilon>0$ such that, given a 1–2-Δ_0 graph G on n vertices, and given the promise that either its minimum-cost 2*-VCSS H has cost n, or its cost is greater than or equal to* $(1+\varepsilon)\,n$*, it is $\mathcal{NP}$-hard to distinguish which of the two cases holds. In particular, it is $\mathcal{NP}$-hard to approximate within* $(1+\varepsilon)$ *the cost of a minimum-cost* 2*-VCSS of a 1–2-Δ_0 graph.*

The next result is a direct application of Theorem 51.10 combined with classical results on metric embeddings.

Theorem 51.11 [18]

For any fixed $p \geq 1$ there exists a constant $\xi > 0$ such that it is $\mathcal{NP}$-hard to approximate within $1 + \xi$ the minimum-cost 2-connected graph spanning a set of n points in the ℓ_p metric in $\mathbb{R}^{\log n}$.

Corollary 51.1

The minimum k-VCSS problem in graphs of maximum degree bounded by some constant is APX-hard and hence does not have a PTAS unless $\mathcal{P} = \mathcal{NP}$.

One can easily modify the proofs of the theorems presented in this section in order to obtain similar inapproximability results for the problem of finding a minimum-cost k-edge-connected subgraph of a k-vertex-connected graph.

51.7 Extensions to Other Related Problems

The results and techniques we discussed in the previous sections can be applied to various related problems.

51.7.1 Pseudo-Approximations and Steiner *k*-VCSS/ECSS

It is not hard to improve the pseudo-approximation result obtained in Theorem 51.2 by modifying the result from Theorem 51.8. We begin with finding a k-ECSSM whose cost is within $1 + \varepsilon$ of the minimum using the result from Theorem 51.8. Then, we can trivially transform this multigraph into a Steiner k-VCSS by placing $k - 1$ Steiner points on each input point (i.e., at the length zero from it) and forming a k-clique of zero cost out of the point and its associated $k - 1$ Steiner points. The cost of the resulting graph is within $1 + \varepsilon$ of the minimum-cost of k-ECSSM for the input set, which, in turn, does not exceed $1 + \varepsilon$ times the minimum-cost k-VCSS on the input set. Such a Steiner k-VCSS can be found in (asymptotically) the same time as required by Theorem 51.8 to find the k-ECSSM, which is significantly better than the result in Theorem 51.2. The same approach works also for Steiner k-ECSS.

51.7.2 Steiner *k*-Connectivity—Real Approximation Schemes

The techniques described in the survey can be also used to derive efficient approximation schemes for Euclidean minimum-cost Steiner k-connectivity. In contrast to the result in Section 51.7.1, our goal is to find a Steiner k-VCSS (or k-ECSS) for a set of points S in $\mathbb{R}^d$ whose cost is at most $1 + \varepsilon$ times the minimum-cost of Steiner k-VCSS (k-ECSS, respectively) for S; so both, the solution found and the optimal solution are allowed to use Steiner points.

The main difficulty with extending the result from Section 51.7.1 to a real PTAS for Steiner k-VCSS/ECSS is that the spanners used in the Structure Theorem III (Theorem 51.7) and in the PTAS from Theorem 51.8 do not include Steiner points. Nevertheless, one can decompose optimal Steiner solutions for k-connectivity and combine this decomposition with the construction of banyans due to Rao and Smith [32]. The case of $k = 2$ is most interesting. Extending the work of Hsu and Hu [30], Czumaj and Lingas [24] showed a new structural characterization of minimum-cost Steiner biconnected graphs that lead to a decomposition of an optimal Steiner solution into minimum Steiner trees. This opened the possibility of using the so-called $(1 + \varepsilon)$-banyans, for the purpose of approximating the Euclidean minimum Steiner tree problem. As the result, Czumaj and Lingas [24] obtain a PTAS for Euclidean minimum-cost Steiner biconnectivity and Euclidean minimum-cost two edge connectivity; the algorithms run in time $\mathcal{O}(n \log n)$ for any constant dimension and ε. For general d and ε, the running time is $n \log n\,(d/\varepsilon)^{\mathcal{O}(d)} + n\,2^{(d/\varepsilon)^{\mathcal{O}(d^2)}} + n\,2^{2^{d^{\mathcal{O}(1)}}}$.

51.7.3 Survivable Networks

Czumaj et al. [27] extended the analysis from previous sections (in particular, Theorem 51.9) to a more general problem of survivable networks. They considered the variant of the *survivable network design problem* in which for a given set S of n points in Euclidean space $\mathbb{R}^d$ and a connectivity requirement function $r : S \to \mathbb{N}$, the goal is to find a minimum-cost graph G on S such that for any pair of points $x, y \in S$, G has $\min\{r(x), r(y)\}$ internally vertex-disjoint paths between x and y. The two most basic (and of largest practical relevance) variants of this problem are those in which $r(x) \in \{0, 1\}$ and when $r(x) \in \{0, 1, 2\}$, for any point $x \in S$.

First, for the simplest case in which $r(x) \in \{0, 1\}$ for any point $x \in S$, that is, for the *Steiner tree problem*,[4] Czumaj et al. [27] designed a randomized algorithm that, for any constant d and any constant ε, in time $\mathcal{O}(n \log n)$ finds a Steiner tree whose cost is at most $(1 + \varepsilon)$ times larger than the minimum. For general d and ε, its running time is $n \log n\,(d/\varepsilon)^{\mathcal{O}(d)} + n\,2^{(d/\varepsilon)^{\mathcal{O}(d^2)}} + n\,2^{2^{d^{\mathcal{O}(1)}}}$.

Next, for the case when $r(x) \in \{0, 1, 2\}$ for any point $x \in S$ (this is the classical problem investigated thoroughly by Grötschel and Monma et al. [2,5,7,11,12]), Czumaj et al. [27] extended algorithm for the Steiner tree problem to design an algorithm that, for any constant d and any constant ε, in time $\mathcal{O}(n \log n)$ finds a graph satisfying all the vertex connectivity requirements and having the cost at most $(1 + \varepsilon)$ times the minimum. When d and ε are allowed to be arbitrary, its running time is $n \log n\,(d/\varepsilon)^{\mathcal{O}(d)} + n\,2^{(d/\varepsilon)^{\mathcal{O}(d^2)}} + n\,2^{2^{d^{\mathcal{O}(1)}}}$.

Finally, essentially the same techniques can be used to obtain a PTAS for the multigraph variant, where the edge-connectivity requirements satisfy $r(x) \in \{0, 1, \ldots, k\}$ and $k = \mathcal{O}(1)$.

All these approximation schemes are randomized, but they can be *derandomized* in a polynomial time.

51.7.4 Finding Low-Cost *k*-VCSS and *k*-ECSS in Planar Graphs

Recently, there has been also a progress in designing approximation schemes for the 2-VCSS and 2-ECSS problem in planar graphs [25,26]. Similarly as for the TSP problem in planar graphs [22,36], the first step toward an efficient approximation scheme has been achieved for unweighted graphs. Czumaj et al. [26] showed that for every positive ε, for a given undirected graph planar G with n vertices, one can find in time $n^{\mathcal{O}(1/\varepsilon)}$ a 2-VCSS (or 2-ECSS) of G whose total number of edges is at most $(1 + \varepsilon)$ times the minimum number of edges in any 2-VCSS (or 2-ECSS, respectively) of G; this gives a PTAS for the unweighted version of the 2-VCSS and 2-ECSS problem in planar graphs. In fact, the approximation scheme provided in Ref. [26] works also for the weighted case, but then the running time becomes $n^{\mathcal{O}(\gamma/\varepsilon)}$, where γ is the ratio of the total edge cost to the optimum solution cost.

Soon after, Berger et al. [25] modified the scheme from Ref. [26] and obtained a *quasi-polynomial time approximation scheme* for the 2-VCSS and 2-ECSS problem in planar graphs. Their algorithm runs in time $n^{\mathcal{O}(\log n \log(1/\varepsilon)/\varepsilon)}$ and finds a 2-VCSS (or 2-ECSS) of G whose total cost is at most $(1 + \varepsilon)$ times the minimum-cost 2-VCSS (or 2-ECSS, respectively) of G. Furthermore, their algorithm can be extended to solve within the same runtime bounds the survivable network design problem in planar graphs in which $r(x) \in \{1, 2\}$ for any vertex.

The underlying techniques developed for the approximation schemes for the 2-VCSS and 2-ECSS problem in planar graphs were surprisingly similar to those used for geometric graphs: a combination of (new) separator theorems with dynamic programming, and then new constructions of *light spanners* for planar graphs. For more details, we refer interested readers to the original papers [25,26].

[4] Note that this variant of the Steiner tree problem is different from the Steiner tree problem considered by Arora [13], for which a PTAS is also known [13,31] (see also Refs. [21,32]). The variant considered in this survey requires that all locations of Steiner points are given in advance (they are the points $x \in S$ with $r(x) = 0$), while in the other variant, all points in $\mathbb{R}^d$ could be used as Steiner points.

51.8 Final Comments

In this chapter, we surveyed recent approximation schemes for various variants of network design problems for geometric graphs. Our main goal was not only to show the result, but also to demonstrate a variety of new techniques developed to coupe with these problems.

51.8.1 Interesting Open Questions

In our context, perhaps the most intriguing open problem for now is whether the minimum-cost 2-VCSS and 2-ECSS problems for planar graphs has a PTAS. We conjecture that this is indeed the case, but so far, the existing techniques seem to be too weak. Further, it would be interesting to see if there is a PTAS for the k-VCSS/ECSS problem in planar graphs for $k = 3, 4$ (note that for $k \geq 5$ no planar graph can be k-vertex-connected).

Another interesting open problem is whether there exists a PTAS for the geometric minimum-cost k-VCSS and k-ECSS problems for very large values of k. The techniques presented in this survey seem to work only for the values of k up to $(\log \log n)^c$ for certain positive constant $c < 1$. What about large values of k?

Finally, and perhaps most importantly, how practical are the methods discussed in this survey? Even though, most probably any direct implementation of the PTAS for k-connectivity problems would be inferior to the existing heuristic implementations discussed (e.g., Refs. [2,5,7,12]), we believe that the techniques presented in this survey when combined with heuristics could lead to significant improvements in practical implementations.

Acknowledgments

We thank Hairong Zhao, Christos Levcopoulos, Sanjeev Arora, and Luca Trevisan for their valuable comments and suggestions. We thank an anonymous referee for pointing out the relation between TST and 2-VCSS in 1-2-graphs.

Research supported in part by NSF ITR grant CCR-0313219 and by VR grant 621-2002-4049.

References

[1] Ahuja, R. K., Magnanti, T. L., Orlin, J. B., and Reddy, M. R., Applications of network optimization, in *Handbooks in Operations Research and Management Science*, *Vol. 7: Network Models*, Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., Eds., Elsevier, North-Holland, Amsterdam, 1995, chap. 1, pp. 1–83.

[2] Grötschel, M., Monma, C. L., and Stoer, M., Design of survivable networks, in *Handbooks in Operations Research and Management Science*, *Vol. 7, Network Models*, Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., Eds., 1995, chap. 10.

[3] Goemans, M. X. and Williamson, D. P., The primal-dual method for approximation algorithms and its application to network design problems, in *Approximation Algorithms for $\mathcal{N}$P-Hard Problems*, Hochbaum, D. S., Ed., PWS Publishing Company, Boston, MA, 1996, chap. 4.

[4] Schrijver, A., *Combinatorial Optimization Polyhedra and Efficiency*, Springer-Verlag, New York, 2003.

[5] Stoer, M., *Design of Survivable Networks*, *Lecture Notes in Mathematics*, Vol. 1531, Springer-Verlag, New York, 1992.

[6] Mihail, M., Shallcross, D., Dean, N., and Mostrel, M., A commercial application of survivable network design: ITP/INPLANS CCS network topology analyzer, *Proc. SODA*, 1996, p. 279.

[7] Monma, C. L. and Shallcross, D. F., Methods for designing communications networks with certain two-connected survivability constraints, *Oper. Res.*, 37(4), 531, 1989.

[8] Du, D. Z. and Hwang, F. K., A proof of the Gilbert–Pollak conjecture on the Steiner ratio, *Algorithmica*, 7, 121, 1992.

[9] Hwang, F. K., Richards, D. S., and Winter, P., *The Steiner Tree Problem*, *Annals of Disc. Math.*, Vol. 53, North-Holland, Amsterdam, 1992.

[10] Gabow, H. N., Goemans, M. X., and Williamson, D. P., An efficient approximation algorithm for the survivable network design problem, *Math. Prog., Ser. B*, 82(1–2), 13, 1998.

[11] Grötschel, M. and Monma, C. L., Integer polyhedra arising from certain network design problems with connectivity constraints, *SIAM J. Disc. Math.*, 3(4), 502, 1990.

[12] Grötschel, M., Monma, C. L., and Stoer, M., Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints, *Oper. Res.*, 40(2), 309, 1992.

[13] Arora, S., Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems, *JACM*, 45(5), 753, 1998.

[14] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, New York, 1979.

[15] Hochbaum, D. S., Ed., *Approximation Algorithms for $\mathcal{NP}$-Hard Problems*, PWS Publishing Company, Boston, MA, 1996.

[16] Kortsarz, G. and Nutov, Z., Approximating k-node connected subgraphs via critical graphs, *SIAM J. Comput.*, 35(1), 247, 2005,

[17] Khuller, S., Approximation algorithms for finding highly connected subgraphs, in *Approximation Algorithms for $\mathcal{NP}$-Hard Problems*, Hochbaum, D. S., Ed., PWS Publishing Company, Boston, MA, 1996, chap. 6.

[18] Czumaj, A. and Lingas, A., On approximability of the minimum-cost k-connected spanning subgraph problem, *Proc. SODA*, 1999, p. 281.

[19] Frederickson, G. N. and JáJá, J., On the relationship between the biconnectivity augmentation and traveling salesman problem, *Theor. Comp. Sci.*, 19(2), 189, 1982.

[20] Cheriyan, J. and Vetta, A., Approximation algorithms for network design with metric costs, *Proc. STOC*, 2005, p. 167.

[21] Mitchell, J. S. B., Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems, *SIAM J. Comput.*, 28(4), 1298, 1999.

[22] Arora, S., Grigni, M., Karger, D., Klein, P., and Woloszyn, A., A polynomial-time approximation scheme for weighted planar graph TSP, *Proc. SODA*, 1998, p. 33.

[23] Czumaj, A. and Lingas, A., A polynomial time approximation scheme for Euclidean minimum cost k-connectivity, *Proc. ICALP*, 1998, p. 682.

[24] Czumaj, A. and A. Lingas, A., Fast approximation schemes for Euclidean multi-connectivity problems, *Proc. ICALP*, 2000, p. 856.

[25] Berger, A., Czumaj, A., Grigni, M., and Zhao, H., Approximation schemes for minimum 2-connected spanning subgraphs in weighted planar graphs, *Proc. ESA*, 2005, p. 472.

[26] Czumaj, A., Grigni, M., Sissokho, P., and Zhao, H., Approximation schemes for minimum 2-edge-connected and biconnected subgraphs in planar graphs, *Proc. SODA*, 2004, p. 489.

[27] Czumaj, A., Lingas, A., and Zhao, H., Polynomial-time approximation schemes for the Euclidean survivable network design problem, *Proc. ICALP*, 2002, p. 973.

[28] Czumaj, A. and Lingas, A., Polynomial time approximation schemes for Euclidean multi-connectivity problems, submitted for publication, 2007.

[29] Czumaj, A. and Lingas, A., Fast approximation schemes for Euclidean biconnectivity problems and for multi-connectivity problems, submitted for publication, 2007.

[30] Hsu, D. F. and Hu, X.-D., On short two-connected Steiner networks with Euclidean distance, *Networks*, 32(2), 133, 1998.

[31] Arora, S., Approximation schemes for $\mathcal{NP}$-hard geometric optimization problems: a survey, *Math. Prog., Series B*, 97(1–2), 43, July 2003.

[32] Rao, S. B. and Smith, W. D., Approximating geometrical graphs via "spanners" and "banyans", *Proc. STOC*, 1998, p. 540.

[33] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B., Eds., *The Traveling Salesman Problem*, Wiley, New York, 1985.

[34] Gudmundson, J., Levcopoulos, C., and Narasimhan, G., Fast greedy algorithms for constructing sparse geometric spanners, *SIAM J. Comput.*, 31(5), 1479, 2002.

[35] Trevisan, L., When Hamming meets Euclid: the approximability of geometric TSP and Steiner Tree, *SIAM J. Comput.*, 30(2), 475, 2000.

[36] Grigni, M., Koutsouias, E., and Papadimitriou, C., An approximation scheme for planar graph TSP, *Proc. FOCS*, 1995, p. 640.

52

Dilation and Detours in Geometric Networks

Joachim Gudmundsson
National ICT Australia Ltd

Christian Knauer
Free University of Berlin

52.1 Introduction

In this chapter we consider geometric networks and two important quality measures of such networks, namely *dilation* and *detour*. A *geometric network* is an undirected graph whose vertices are points in d-dimensional space, and the edges are straight-line segments connecting the vertices. The sites are usually located in the Euclidean plane, but other metrics and higher dimensions are also common. Geometric networks arise in many applications. Road networks, railway networks, telecommunication, pattern matching, bioinformatics—any collection of objects in space that have some connections between them can be modeled as a geometric network.

The weight of an edge $e = (u, v)$ in a geometric network $G = (S, E)$ on a set S of n points is the (usually Euclidean) distance between u and v, which we denote by $d(u, v)$. The *graph distance* $d_G(u, v)$ between two vertices $u, v \in S$ is the length of the shortest path in G connecting u to v.

This chapter will consider the problem of designing a "good" network and the dual problem, that is, evaluating how "good" a given network is. When designing a network for a given set S of points, several criteria have to be taken into account. In particular, in many applications it is important to ensure a fast connection between every pair of points in S. For this it would be ideal to have a direct connection between every pair of points; the network would then be a complete graph. In most applications, however, this is unacceptable due to the high costs. This leads to the concepts of dilation and detour of a graph, which we define next.

The *dilation* or *stretch factor* of G, denoted $\Delta(G)$, is the maximum factor by which the graph distance d_G differs from the geometric distance d between every pair of vertices.

Definition 52.1

Let S be a set of points in $\mathbb{R}^d$, let $M = (\mathbb{R}^d, d_M)$ be a metric space on $\mathbb{R}^d$ and let G be a graph in M with vertex set S. For any two vertices $x, y \in S$ let $d_G(x, y)$ be the infimum length of all paths connecting x to y

in G. We call

$$\Delta_M^G(x, y) = \frac{d_G(x, y)}{d_M(x, y)}$$

the M-dilation between x and y in G, and $\Delta_M(G) = \sup_{x,y \in S} \Delta_M^G(x, y)$ the M-dilation of G.

A graph $G = (S, E)$ with $\Delta_M(G) \leq t$ is said to be a *t-spanner* of S.

The notion of dilation can be generalized to arbitrary connected sets $P \subset \mathbb{R}^d$. This measure is called *detour* and compares the length of a shortest path inside P between any two points $x, y \in P$ with their distance measured, for example, in the Euclidean metric on $\mathbb{R}^d$.

Definition 52.2

Let $P \subset \mathbb{R}^d$ be a connected set, and $M = (\mathbb{R}^d, d_M)$ be a metric space on $\mathbb{R}^d$. For any two points, $x, y \in P$ let $d_P(x, y)$ be the infimum length of all curves connecting x to y that are contained in P. We call

$$\delta_M^P(x, y) = \frac{d_P(x, y)}{d_M(x, y)}$$

the M- detour between x and y in P, and $\delta_M(P) = \sup_{x,y \in P} \delta_M^P(x, y)$ the M-detour of P.

In some sense the detour measures how much two metric spaces on $\mathbb{R}^d$—namely M and $\mathbb{R}^d$ with the shortest-path metric induced by P—resemble each other.

In this chapter we will mainly consider the case $M = \mathbb{E}^d$, the d-dimensional Euclidean space. We then write $\Delta(G)$ and $\delta(P)$ instead of $\Delta_{\mathbb{E}^d}(G)$ and $\delta_{\mathbb{E}^d}(P)$, respectively. Whenever we speak about the dilation or detour without specifying M, we refer to the case $M = \mathbb{E}^d$.

The chapter is organized as follows. In Section 52.2 we give an overview of the construction of t-spanners. Section 52.3 briefly considers the dual problem, namely computing the dilation of a given graph. Then in Section 52.4 we turn our attention to the problem of computing the detour. Finally, in Section 52.5, we end the chapter by looking at structures with small dilation.

52.2 Constructing *t*-Spanners

The problem considered in this section is the construction of t-spanners given a set S of n points in $\mathbb{R}^d$ and a positive real value $t > 1$. The aim is to compute a good t-spanner for S where the quality measures are:

Size. The number of edges in the graph.

Degree. The maximum number of edges incident to a vertex.

Weight. The weight of an Euclidean network G is the sum of the edge weights.

Spanner diameter (or simply diameter). Defined as the smallest integer d such that for any pair of vertices u and v in S, there is a t-path in the graph (a path of length at most $t \cdot |uv|$) between u and v containing at most d edges.

There are trade-offs between different quality measures, for example, between the degree and the diameter [1]; a graph with constant degree will have diameter $\Omega(\log n)$. A further example is the trade-off between the diameter and the weight [2], that is, if the diameter of a Euclidean graph G is bounded by $O(\log n)$ then the weight of G is $\Omega(wt(MST(S))\frac{\log n}{\log \log n})$, where $wt(MST(S))$ denotes the weight of the minimum spanning tree of S. Finally, there is also an $\Omega(n \log n)$ time lower bound in the algebraic computation tree model for computing any t-spanner for a given set of points S [3].

The most well-known t-spanners can be divided into three groups: Θ-graphs, WSPD-graphs, and greedy-graphs. In the following sections we give the main idea of each of these, together with the known bounds. Throughout this section it will be assumed that the set of input points is given in d-dimensional Euclidean space. For a more detailed description of the construction of t-spanners see the extensive and thorough work by Narasimhan and Smid [4].

52.2.1 The Θ-Graph

The Θ-graph was discovered independently by Clarkson [5] and Keil [6]. Keil considered the graph in two dimensions while Clarkson extended his construction to also include three dimensions. Althöfer et al. [7] defined the Θ-graph for higher dimensions and Ruppert and Seidel [8] improved the construction time to $O(n \log^{d-1} n)$. The general approach is stated below. Note that it is possible to cover $\mathbb{E}^d$ by k simplicial cones of angular diameter θ, where $k = O(1/\theta^{d-1})$ as defined in the algorithm.

Algorithm Θ-GRAPH(S, t)

```
Set $k := 2d!\left\lceil \sqrt{\frac{2(d-1)}{1-\cos\theta}} \right\rceil^{d-1}$ such that $t = \frac{1}{\cos\theta - \sin\theta}$ for $\theta = 2\pi/k$.
Set $E := \emptyset$.
for each point $u \in S$
    Consider $k$ cones $C_1, \ldots, C_k$ with angular diameter $\theta$ and apex at $u$ that cover $\mathbb{E}^d$.
    for each cone $C_i$
        Find the point $v$ within $C_i$ whose orthogonal projection onto the bisector of $C_i$ is closest to $u$.
        Add $(u, v)$ to $E$.
return $G = (S, E)$.
```

A similar construction was already defined by Yao [9] in 1982, with the difference that for every point u and every cone C_i, u is connected to the closest point in C_i. Defining the edges as in the Θ-graph algorithm has the advantage of faster computation.

Theorem 52.1

The Θ-graph is a t-spanner of S for $t = \frac{1}{\cos\theta - \sin\theta}$ with $O(\frac{n}{\theta^{d-1}})$ edges and can be computed in $O(\frac{n}{\theta^{d-1}} \log^{d-1} n)$ time using $O(\frac{n}{\theta^{d-1}} + n \log^{d-2} n)$ space.

Even though the "out-degree" of each vertex is bounded by k, the "in-degree" could be linear. Also, in worst case the weight and the diameter of the Θ-graph can be $\Omega(n \cdot wt(MST(S)))$ and $n-1$, respectively. However, there are several variants of the Θ-graph that improve these bounds.

Sink-Spanners

To obtain a spanner with constant degree one can use the construction of sink-spanners by Arya et al. [1]. The basic idea is as follows. Start with a Θ-graph that is a $\sqrt{t}$-spanner. Direct all the edges such that the out-degree is bounded by a constant for every vertex. To handle the vertices with high in-degree, replace each high degree node q and its adjacent neighbors, that is, the star centered at q, with a bounded degree $\sqrt{t}$-sink-spanner. A $\sqrt{t}$-sink-spanner is a directed graph where each point has a directed $\sqrt{t}$-spanner path to the center q. This is done in a way that may increase the dilation by a factor of $\sqrt{t}$, thus resulting in a t-spanner with degree $O\left(\frac{1}{(t-1)^{2d-2}}\right)$.

Theorem 52.2

The sink spanner is a t-spanner of S for $t = \frac{1}{(\cos\theta - \sin\theta)^2}$ with $O(\frac{n}{\theta^{d-1}})$ edges and can be computed in $O(\frac{n}{\theta^{d-1}} \log^{d-1} \frac{n}{\theta^{d-1}})$ time using $O(\frac{n}{\theta^{d-1}} + n \log^{d-2} n)$ space.

The transformation from a directed $\sqrt{t}$-spanner with bounded out-degree to a t-spanner with bounded degree is called a sink-spanner transformation.

Skip-List Spanners

The idea is to generalize skip-lists [10] and apply them to the construction of spanners. Construct a sequence of subsets, as follows: Let $S_1 = S$. Let $i > 1$ and assume that we already have constructed the subset S_i. For each point in S_i, flip a fair coin. The set S_{i+1} is defined as the set of all points of S_i whose coin flip produced heads. The construction stops if $S_{i+1} = \emptyset$. We have $\emptyset = S_{h+1} \subset S_h \subseteq S_{h-1} \subseteq \cdots \subseteq S_1 = S$. It holds that $h = O(\log n)$ with high probability and that $\sum_{i=1}^{h} |S_i| = O(n)$ with high probability. For each $1 \leq i \leq h$, construct a Θ-graph $G(S_i)$. The union of the graphs $G(S_1), \ldots, G(S_h)$ is the skip-list spanner G. The skip-list spanner is a t-spanner having $O(n)$ edges and $O(\log n)$ spanner diameter with high probability.

Ordered* $\boldsymbol{\Theta}$*-Graphs

A recent modification of the Θ-graph by Bose et al. [11] is the so-called *Ordered* Θ*-graph* which considers the order in which the points of S are processed, that is, the graph is built incrementally by inserting and processing each point in some predefined order. When a new point is processed it only considers the points in the graph that have already been processed. They show an ordering that guarantees that the degree is bounded by $O(k \log n)$ and that a random order gives a t-spanner for which the diameter is bounded by $O(\log n)$ with high probability.

Gap-Greedy

The final variant is a combination of the Θ-graph and a greedy approach. A set of directed edges is said to satisfy the *gap* property if the sources of any two edges in the set are separated by a distance that is at least proportional to the length of the shorter of the two edges. Chandra et al. [12] showed that any directed graph G that fulfills the *gap* property has weight $O(\log n \cdot wt(MST(S)))$. However, the gap property is limited in power. Lenhof et al. [13] showed that there exists a graph that satisfies the gap property and has weight $\Omega(\frac{\log n}{\log \log n} \cdot wt(MST(S)))$.

Using the above idea, Arya and Smid [14] proposed an algorithm that uses the gap property to decide if an edge should be added to the t-spanner graph or not. They consider pairs of points in order of increasing distance, adding an edge (p, q) if and only if it does not violate the gap property.

Theorem 52.3

Let $t = 1/(\cos\theta - \sin\theta - 2w)$ for some real numbers $0 < \theta < \pi/4$ and $0 \leq w < (\cos\theta - \sin\theta)/2$. The gap-greedy algorithm produces a t-spanner G of S in time $O(n/\theta^{d-1} \log^d n)$ such that each vertex has degree $O(1/\theta^{d-1})$ and weight $O(\frac{1}{\theta^{d-1}} \cdot (1 + \frac{1}{w}) \log n \cdot wt(MST(S)))$.

52.2.2 The Well-Separated Pair Decomposition-Graph

The well-separated pair decomposition (WSPD) was developed by Callahan and Kosaraju [15]. A detailed description of the WSPD can be found in Chapter 53 by Smid in this handbook. The WSPD-graph was first described by Callahan and Kosaraju in [16] but similar ideas were used earlier by Salowe [17,18] and Vaidya [19–21].

```
Algorithm WSPD-GRAPH($S$, $t$)
$E' := \emptyset$
$G' := (S, E')$.
$\{A_1, B_1\}, \ldots, \{A_m, B_m\} \leftarrow$ the well-separated pair decomposition of $S$ w.r.t. $s = \frac{4(t+1)}{(t-1)}$.
for each well-separated pair $\{A_i, B_i\}$
    Let $a_i$ and $b_i$ be arbitrary points in $A_i$ and $B_i$ respectively.
    Add $(a_i, b_i)$ to $E'$.
return $G' = (S, E')$.
```

The following theorem summarizes the properties.

Theorem 52.4

The WSPD-graph is a t-spanner for S with $O(s^d \cdot n)$ edges and can be constructed in time $O(s^d n + n \log n)$, where $s = 4(t+1)/(t-1)$.

There are modifications that can be made to obtain bounded diameter or bounded degree.

Bounded Diameter

Arya et al. [22] showed how the construction algorithm can be modified such that the diameter of the graph is bounded by $2 \log n$. In the basic construction of a WSPD-graph a graph is constructed by adding an edge for every well-separated pair in the WSPD. Instead of selecting an arbitrary point in each well-separated set, they choose a representative point by a search in the fair-split tree (see Chapter 53), that is, for a node u in the split tree, follow the path down the tree by always choosing the larger subtree. The point stored at the leaf in which the path ends is the representative point for u. This approach guarantees that the diameter of the constructed t-spanner is bounded by $2 \log n$.

Bounded Degree

The main problem in the construction of the WSPD-graph is that a single point v can be part of many well-separated pairs and each of the pairs generates an edge with an endpoint at v. Arya et al. [1] suggest to keep only the shortest edge for each cone direction, thus combining the Θ-graph approach with the WSPD-graph. The resulting spanner has bounded "out-degree" and by applying the sink-spanner transformation, a t-spanner of degree $O(\frac{1}{(t-1)^{2d-1}})$ is obtained.

52.2.3 The Greedy-Graph

The greedy algorithm was first presented in 1989 by Bern. Althöfer et al. [7] gave the first theoretical bounds and since then the greedy algorithm has been subject to considerable research [12,23–28].The graph constructed using the greedy algorithm is called a greedy-graph and the general approach is given below.

```
Algorithm GREEDY-GRAPH($S$, $t$)
Construct the complete graph of $S$, denoted $G = (S, E)$.
$E' := \emptyset$
$G' := (S, E')$.
for each edge $(u, v) \in E$ in order of increasing weight
        if SHORTESTPATH($G'$, $u$, $v$) $> t \cdot d_G(u, v)$
                Add $(u, v)$ to $E'$.
return $G' = (S, E')$.
```

Chandra et al. [12] proved that the maximum degree of the graph is bounded by a constant. The running time of the naïve implementation of GREEDY-GRAPH considered in their paper is $O(n^3 \log n)$. Das, Narasimhan and Salowe [26] showed that the greedy-spanner fulfills the so-called *leapfrog property*. A set of undirected edges E is said to satisfy the t-leapfrog property, if for every $k \geq 2$, and for every possible sequence $\{(p_1, q_1), \ldots, (p_k, q_k)\}$ of pairwise distinct edges of E,

$$t \cdot |p_1 q_1| < \sum_{i=2}^{k} |p_i q_i| + t \cdot \left(\sum_{i=1}^{k-1} |q_i p_{i+1}| + |p_k q_1| \right)$$

Using the leapfrog property it is possible to bound weight of the graph. Das and Narasimhan [25] made a breakthrough in 1994 when they showed that a modified greedy graph could be constructed in $O(n \log^2 n)$ time. They detailed how to use clustering to speed up shortest path queries, by showing that approximate shortest path queries suffice to produce sparse spanners. However, their algorithm was not efficient as the clusters were not maintained efficiently and had to be frequently rebuilt. This problem was solved by Gudmundsson et al. [27], who developed techniques to efficiently perform clustering. Das and Narasimhan [25] proved that the edges in the greedy graph satisfies the so-called leapfrog property and showed that any graph satisfying this property has weight $O(wt(MST(S)))$. A set of undirected edges E is said to satisfy the t-leapfrog property, if for every $k \geq 2$, and for every possible sequence $\{(p_1, q_1), \ldots, (p_k, q_k)\}$ of pairwise distinct edges of E,

$$t \cdot |p_1 q_1| < \sum_{i=2}^{k} |p_i q_i + t \cdot \left(\sum_{i=1}^{k-1} |q_i p_{i+1}| + |p_k q_1| \right)$$

A complete proof can be found in Chapter 14 in the book by Narasimhan and Smid [4]. The following theorem summarizes the known bounds.

Theorem 52.5

The greedy graph is a t-spanner of S with $O(\frac{n}{(t-1)^d}\log(\frac{1}{t-1}))$ *edges,* $O(\frac{1}{(t-1)^d}\log(\frac{1}{t-1}))$ *maximum degree and total weight* $O(\frac{1}{(t-1)^{2d}}\cdot wt(MST(S)))$*, and can be computed in time* $O(\frac{n}{(t-1)^{2d}}\log n)$.

52.2.4 Experimental Studies

The first experimental study of the construction of t-spanners was performed by Navarro and Paredes [30], who presented four heuristics for point sets in high-dimensional metric space ($d = 20$) and showed by empirical methods that the running time was $O(n^{2.24})$ and the number of edges in the produced graphs was $O(n^{1.13})$. In Ref. [31], Sigurd and Zachariasen considered the problem of constructing a minimum-weight t-spanner of a given graph, but they only considered sparse graphs of small size, that is, graphs with at most 64 vertices and with average vertex degree 4 or 8. In the case where the input points are given in the Euclidean plane, an extensive study of the main algorithms presented in Sections 52.2.1–52.2.3 was performed by Farshi and Gudmundsson [32].

52.3 Computing the Dilation of a Graph

The previous section considered the problem of constructing a graph for a given point set. In this section the dual problem is considered, that is, given a graph G compute $\Delta(G)$.

The problem of calculating the dilation of a given geometric graph can be solved by computing All-Pairs-Shortest-Paths of G. Running Dijkstra's algorithm—implemented using Fibonacci heaps—gives the dilation of G in time $O(mn + n^2 \log n)$ using linear space, where m is the number of edges in G. For a long time there were no considerable improvements but in 2002, Langerman et al. [33] and Agarwal et al. [34] showed the first subquadratic bounds for certain types of graphs. They proved that the dilation of a planar polygonal path can be computed in $O(n \log n)$ expected time. The algorithm can be generalized to planar trees and cycles, with a randomized expected running time of $O(n \log^2 n)$, or $O(n \log^{O(1)} n)$ worst-case running time. Agarwal et al. [35] also showed that in three dimensions one can compute the dilation of a path, cycle, or tree in $O(n^{4/3+\epsilon})$ in randomized expected time. More details about their construction can be found in Section 52.4.2.

Eppstein and Wortman [36] presented an $O(n \log n)$-time algorithm for evaluating the dilation when the input graph G is a star. Computing the shortest path between two points in a star obviously takes constant time, their idea is to identify $O(n)$ candidate pairs and prove that the pair deciding the dilation of G is among these pairs. The point pairs are identified using two techniques, each generating $O(n)$ pairs.

Assume that (x, y) is a pair of points in G with dilation $\Delta(G)$.

In the case when the dilation of G is high, that is, greater than 3, then it holds that y is one of x's k nearest neighbors, for a constant k. The k nearest neighbors of every point in V may be reported in time $O(kn \log n)$ using the algorithm by Vaidya [20]. So the process of identifying the $O(n)$ candidate point pairs takes $O(n \log n)$ time.

In the case when the dilation of G is low, that is, smaller than or equal to 3, then it holds that x and y must have almost the same distance to the center of G. Assume that the vertices of V are sorted with respect to their distance from the center of G, $\langle v_1, \ldots, v_n \rangle$, and that $x = v_i$ and $y = v_j$. It holds that $|i - j| \leq \ell$, where ℓ is a constant, and thus identifying $O(n)$ candidate point pairs requires $O(n \log n)$ time in this case.

52.3.1 Approximating the Dilation

For general geometric graphs it seems unavoidable to test all the $\binom{n}{2}$ pairs of vertices that may decide the dilation of the graph. However, in the case when it suffices to approximate the dilation, this bound is no longer correct. Narasimhan and Smid [37] showed that $O(n/\epsilon^d)$ pairs of vertices are sufficient to test to give a good approximation. Their algorithm is very simple and it is stated below. It is assumed that an algorithm $\text{ASP}_c(p, q, G)$ is given that takes a graph G and two vertices p and q as input and computes a c-approximation of $\Delta_G(p, q)$, where $\Delta_G(p, q) = \frac{d_G(p,q)}{d(p,q)}$. Denote by $T(n, m, k)$ the running time of ASP_c, when given (i) a graph having n vertices, m edges, and (ii) a sequence of k c-approximate shortest path queries.

The algorithm takes a Euclidean graph $G = (V, E)$ and a real constant $\epsilon > 0$ as input. A well-separated pair decomposition of V is computed with separation constant $4(1 + \epsilon)/\epsilon$. For each well-separated pair $\{A_i, B_i\}$ two arbitrary points $a_i \in A_i$ and $b_i \in B_i$ are chosen, and the dilation of a_i and b_i is estimated by taking the ratio between $\text{ASP}_c(a_i, b_i, G)$ and $d(a_i, b_i)$. The maximum over all the values is returned.

The main theorem can now be stated.

Theorem 52.6

Let G be a Euclidean graph in $\mathbb{E}^d$ and let ϵ be a real constant such that $0 < \epsilon \leq 3$, one can compute a $((1 + \epsilon)c)$-approximate dilation of G in time

$$O(n \log n) + T(n, m, n/\epsilon^d)$$

Using known data structures to answer (approximate) distance queries together with Theorem 52.6 gives an $O(n \log n)$ time $(1 + \epsilon)$-approximation algorithm for paths, cycles, and trees, an $O(n\sqrt{n})$ time $(1+\epsilon)$-approximation algorithm for plane graphs [38], an $O(m+n(t^5/\epsilon^2)^d (\log n+(t/\epsilon)^d))$ time $(1+\epsilon)$-approximation algorithm for t-spanners [39,40], and an $O(mn^{1/\beta} \log^2 n)$ expected time $O(2\beta(1 + \epsilon)^2)$-approximation algorithm for any Euclidean graph [41].

Note that any algebraic computation tree algorithm that computes a c-approximate dilation of a path or a cycle has running time $\Omega(n \log n)$ [37].

52.4 Detour

When a geometric network G models an urban street system, the dilation is not necessarily an appropriate measure to assess the quality of G. Since houses are spread everywhere along the streets, one has to take into account not only the vertices of G but also all the points on its edges, that is, consider the detour of G. The detour is also of particular interest in various other applications:

- Analyzing on-line navigation strategies often involves estimating the detour of curves: The length of a path created by some robot must be compared to the shortest path connecting two points [42].
- When comparing the Fréchet distance $F(P, Q)$ between two curves P,Q of detour at most κ with their Hausdorff distance $H(P, Q)$ (see Ref. [43] for the definition of F and H) it turns out that (under some additional technical condition) $F(P, Q) \leq (1 + \kappa)H(P, Q)$, while no such bound is known for general curves [44,45].

In Section 52.4.1 we describe properties of pairs of points with maximum detour for various scenarios, and in Section 52.4.2 we give algorithms for computing the detour in these scenarios. All the algorithms exploit the structural properties described earlier. The problem of constructing graphs of small detour that contain a prescribed finite point set will be considered in Section 52.4.3.

52.4.1 Structural Properties

We describe properties of pairs of points with maximum detour for the following scenarios:

- P is a simple polygonal curve (possibly closed), or a simple tree in $\mathbb{R}^2$,
- P is a simple polygon in $\mathbb{R}^2$, and
- P is a geometric graph in $\mathbb{R}^2$.

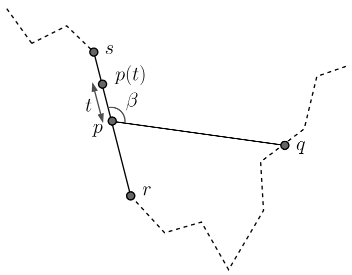


FIGURE 52.1 The detour $\delta^P(p(t), q)$ is larger than $\delta^P(p, q)$.

As already mentioned, the case of the detour of a planar polygonal chain P is of particular interest in various applications. The problem of (approximately) computing $\delta(P)$ in that setting was first addressed in Ebbers-Baumann et al. [46]. The proposed algorithm exploits several structural properties of the problem.

Consider for instance an edge e of P with endpoints r, s, and let q be a fixed point on P, such that $d_P(q, s) > d_P(q, r)$, cf., Figure 52.1. The function $\delta^P(., q)$ takes on a unique maximum on e, and if $\beta := \cos\angle(q, p, s) = -\frac{||p-q||}{d_P(q,r)}$ holds, then this maximum is attained at p.

To see this, let us assume that $0 < \beta < \pi$. For $-||p - r|| \le t \le 0$ let $p(t)$ be the point on $\overline{rp}$ that has distance $|t|$ to p, and for $0 \le t \le ||p - s||$ let $p(t)$ be the point on $\overline{ps}$ that has distance $|t|$ to p. We have

$$f(t) := \delta^P(p(t), q) = \frac{d_P(p(t), q)}{||p(t) - q||} = \frac{t + d_P(p, q)}{\sqrt{t^2 + ||p - q||^2 - 2t||p - q|| \cos\beta}}$$

Since $||p - q|| \cos\beta + d_P(p, q) > 0$, the derivative of $f(t)$ has a positive denominator and its numerator has the same sign as

$$n(t) := ||p - q|| \frac{||p - q|| + d_P(p, q) \cos\beta}{||p - q|| \cos\beta + d_P(p, q)} - t$$

Thus, if $||p - q|| + d_P(p, q) \cos\beta$ is positive (resp. negative), the detour can be increased by moving p toward s (resp. r).

Another crucial property is that there is always a pair of points (p', q') attaining the detour of a simple polygonal chain P, such that p' and q' are *covisible*, that is, $\overline{p'q'} \cap P = \{p', q'\}$. This can be seen as follows: Let $p', q' \in P$ attain the detour of P, that is, $\delta^P(p, q) = \delta(P)$, and $p = p_0, \dots, p_k = q$ be the points of P intersected by the segment $s = \overline{pq}$, ordered by their appearance on s, cf. Figure 52.2.

Then

$$\delta^P(p, q) = \frac{d_P(p, q)}{||p - q||} \le \frac{\sum_{i=0}^{k-1} d_P(p_i, p_{i+1})}{\sum_{i=0}^{k-1} ||p_i - p_{i+1}||} \le \max_{0 \le i < k} \frac{d_P(p_i, p_{i+1})}{||p_i - p_{i+1}||} = \max_{0 \le i < k} \delta^P(p_i, p_{i+1})$$

that is, some covisible pair p_i and p_{i+1} attains the detour of P.

We can summarize the above discussion in the following lemma.

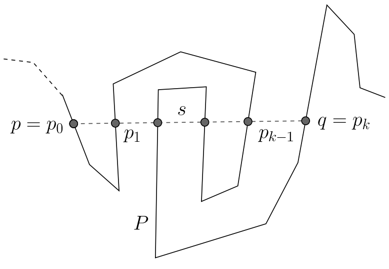


FIGURE 52.2 There is always a covisible pair of points attaining the detour.

Lemma 52.1 [46]

The detour of a simple polygonal chain P in the plane is attained by a pair of points (p, q) on P, where p is a vertex of P, and p and q are covisible.

Similar results were obtained for various other cases. Lemma 52.1 was generalized by Agarwal et al. [35] to the case of simple trees in the plane, and by Ebbers-Baumann et al. [47] it is shown that Lemma 52.1 also holds for simple polygons in the plane.

Lemma 52.2 [48]

For any metric space M on $\mathbb{R}^2$ the M-detour of a simple polygon P in the plane with $\delta_M(P) > 1$ is attained by a pair of points (p, q) on the boundary of P, with the property that $\overline{pq} \cap P = \{p, q\}$, and at least one of p, q is a vertex of P.

For the Euclidean metric, one can even show that every pair of points (p, q) on the boundary that attains the detour of any nonconvex simple polygon P must have the property that $\overline{pq} \cap P = \{p, q\}$.

It is easy to see that the detour of a *closed* simple polygonal curve is *not necessarily* attained at a vertex. Still, a somewhat weaker property was shown by Agarwal et al. [35] for this case.

Lemma 52.3 [35]

The detour of a closed, simple polygonal curve P of length ℓ in the plane is attained by a pair (p, q) of points of P, such that, either one of them is a vertex of P, or $d_P(p, q) = \ell/2$.

Moreover, Ebbers-Baumann et al. [47] observed that it is still the case, that a covisible pair of points attains the detour of P. This is in fact true for arbitrary connected simple straight-line graphs in the plane:

Lemma 52.4 [47]

The detour of a connected simple straight-line graph P in the plane is attained by a pair of covisible points of P.

Note that the *dilation* of a geometric graph is not necessarily attained at a covisible pair of vertices. Moreover most of these properties fail to hold in higher dimensions. The detour of a polygonal curve in $\mathbb{R}^3$ for instance is not necessarily attained at a vertex of the chain, cf. [35].

52.4.2 Algorithmic Questions

Based on the earlier work of Narasimhan and Smid [37], Grüne [49] has shown that there is an $\Omega(n \log n)$ lower bound in the algebraic decision tree model for computing the *dilation* of a monotone and hence simple planar polygonal curve. Unfortunately, for the problem of computing the detour of such curves, no nontrivial lower bound is known.

However, as was shown by Agarwal et al. [35], computing the detour of a three-dimensional polygonal path is as hard as Hopcroft's problem: Given a set L of n lines in $\mathbb{R}^2$ and a set Q of n points in $\mathbb{R}^2$, decide whether any line of L contains any point of Q.

The idea is to reduce an instance of Hopcroft's problem to the problem of computing the detour of a three-dimensional path. To this end, a three-dimensional path $P_{L,Q}$ is built in such a way that $P_{L,Q}$ has infinite detour (i.e., it self-intersects), iff any line of L contains any point of Q. Using techniques developed by Erickson [50] the construction is then modified to cover the case where it is known in advance that the input chain is not self-intersecting.

The construction shows that, if there is an algorithm to compute $\delta(P)$ for a simple polygonal chain P on n vertices in $\mathbb{R}^3$ in $T(n)$ time, then Hopcroft's problem can be solved in $O(n \log n + T(n))$ time. There is an abundance of evidence that suggests that Hopcroft's problem has an $\Omega(n^{4/3})$ lower bound [50] in any reasonable model of computation.

On the positive side, for arbitrary connected plane graphs P on n vertices the detour can be computed in $O(n^2)$ time in the following way: Compute the shortest-path distance for all pairs of vertices of P. Since

P is planar this can be done in $O(n^2)$ time, cf. [51]. For every pair of edges e, f of P, compute the detour $\delta^P(e, f) = \max\{\delta^P(x, y) \mid x \in e, y \in f\}$. This can be done in constant time per pair, since there are only four combinatorial different types of shortest paths going from points on e to points on f.

The structural properties shown in the previous section can be exploited to obtain faster algorithms in some cases. The case where P is a planar polygonal chain without self-intersections was first studied by Ebbers-Baumann et al. [46], where the following result is shown:

Theorem 52.7 [46]

Let P be a simple polygonal chain on n vertices in the plane, and let ϵ be a positive constant. In $O(n \log n)$ time a pair of points (p, q) on P can be computed, such that $\delta(P) \leq (1 + \epsilon)\delta^P(p, q)$.

The problem of exactly computing $\delta(P)$ for a simple polygonal chain P was independently considered by Langermann et al. and Agarwal et al. [33,34] (see also Ref. [35] for a combined version).

The approach in Ref. [35] is as follows: First, a deterministic $O(n \log n)$ time algorithm for the decision problem is developed, that is, an algorithm that decides on input (P, κ) whether $\delta(P) \leq \kappa$. Note that according to Lemma 52.1 $\delta(P) \leq \kappa$ iff $\delta^P(q, p) \leq \kappa$ for all *vertices* $p \in P$ and all points $q \in P$.

This problem can be restated in a form that makes it amenable to range-searching techniques: Let p_0 be the first point of P. For a point $p \in P$, define the *weight* of p to be $\omega(p) = d_P(p, p_0)/\kappa$. Let C denote the cone $z = \sqrt{x^2 + y^2}$ in $\mathbb{R}^3$, and map each vertex $p = (p_x, p_y) \in P$ to the cone $C_p = C + (p_x, p_y, \omega(p))$, that is, translate the apex of C (i.e., the origin) to the point $(p_x, p_y, \omega(p))$. If C_p is regarded as the graph of a bivariate function, which will also be denoted by C_p, then for any point $x \in \mathbb{R}^2$, $C_p(x) = |xp| + \omega(p)$. Map a point $q = (q_x, q_y) \in P$ to the point $\hat{q} = (q_x, q_y, \omega(q))$ in $\mathbb{R}^3$, (cf. Figure 52.3). Now, for any point $q \in P$ and a vertex $p \in P$ that lies "before" q on P (in the sense that $d_P(p, q) = d_P(p_0, q) - d_P(p_0, p)$), $\delta^P(q, p) \leq \kappa$ if and only if $\hat{q}$ lies below the cone C_p:

$$\delta^P(q, p) \leq \kappa \Leftrightarrow \frac{d_P(p_0, q) - d_P(p_0, p)}{|qp|} \leq \kappa \Leftrightarrow \frac{d_P(p_0, q)}{\kappa} \leq |qp| + \frac{d_P(p_0, p)}{\kappa} \Leftrightarrow \omega(q) \leq C_p(q)$$

That is, $\delta^P(q, p) \leq \kappa$ iff $\hat{q}$ lies below the cone C_p, and consequently $\delta(P) \leq \kappa$ iff the polygonal chain $\hat{P} = \{\hat{p} \mid p \in P\}$ lies below the lower envelope of the cones $\{C_p \mid p \text{ is a vertex of } P\}$. By exploiting the covisibility property from Lemma 52.1, this condition can be verified in $O(n \log n)$ deterministic time.

Using a randomized technique of Chan [52] or parametric search [53], the decision procedure is then turned into an algorithm for actually computing $\delta(P)$ (parametric search incurs a polylogarithmic overhead).

Theorem 52.8 [35]

Let P be a simple polygonal chain on n vertices in the plane. There is

- *a deterministic algorithm to decide in $O(n \log n)$ time whether $\delta(P) \leq \kappa$, for any $\kappa > 0$,*
- *a randomized algorithm to compute $\delta(P)$ in $O(n \log n)$ expected time, and*
- *a deterministic algorithm to compute $\delta(P)$ in $O(n \log^{O(1)} n)$ time.*

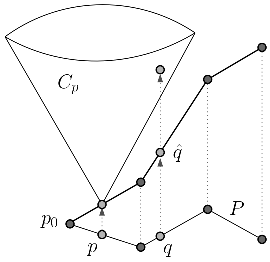


FIGURE 52.3 The chain P lifted to $\mathbb{R}^3$.

Using appropriate recursive partitioning schemes—based on Lemma 52.3 or the variant of Lemma 52.1 for trees—similar techniques can be applied to the case where P is a simple planar tree or cycle.

Theorem 52.9 [35]

Let P be a simple, closed polygonal chain or a plane tree on n vertices in the plane. There is a randomized algorithm to compute $\delta(P)$ in $O(n \log^2 n)$ expected time.

As shown by Grüne et al. [46], the approach can be modified to handle the case where P is a simple polygon in the plane. This yields an efficient approximation algorithm.

Theorem 52.10 [48,49]

Let P be a simple polygon on n vertices in the plane, and let ϵ be a positive constant. In $O(n \log n)$ time a pair of points (p, q) on the boundary of P can be computed, such that $\delta(P) \leq (1 + \epsilon)\delta^P(p, q)$.

The fastest known algorithm for computing $\delta(P)$ for a simple polygon P *exactly* is similar to the brute-force approach described at the beginning of this section. Of course, Lemma 52.2 plays a crucial role here. Moreover, the shortest-path computation is more involved since shortest-geodesic paths inside P have to be computed.

Theorem 52.11 [48]

Let P be a simple polygon on n vertices in the plane. There is a deterministic algorithm to compute $\delta(P)$ in $O(n^2)$ time.

As already mentioned it is no longer true that the detour is attained at a vertex of P, when P is a simple polygonal chain in $\mathbb{R}^3$. This makes the three-dimensional algorithm considerably more complicated, and less efficient, than its two-dimensional counterpart.

Theorem 52.12 [35]

Let P be a simple polygonal chain on n vertices in $\mathbb{R}^3$. There is a randomized algorithm to compute $\delta(P)$ in $O(n^{16/9+\epsilon} \log n)$ expected time for any $\epsilon > 0$.

52.4.3 Low Detour Embeddings of Point Sets

Besides computing the detour of given graphs, the problem of constructing plane graphs of small detour that contain a given finite point set was also investigated.

Definition 52.3 (Detour of a point set)

The detour $\dot{\delta}(P)$ of a finite point set P in the plane is the smallest possible detour of any finite plane graph that contains all points of P, that is,

$$\dot{\delta}(P) := \inf_{\substack{P \subset G \\ G \text{ finite, plane}}} \delta(G)$$

Even for a point set P of size three, computing $\dot{\delta}(P)$ is a nontrivial task. For the *dilation* the optimum solution must be a triangulation, since an optimal solution only contains straight edges, and adding edges never increases the dilation. Still, it is not known how to efficiently compute the triangulation that minimizes the dilation of a given point set.

As a consequence of Lemmas 52.4 and 52.3, the detour of any *rational* point set P is bounded by two, since it can be embedded into a square grid, that is, $\dot{\delta}(P) \leq 2$ for all $P \subseteq \mathbb{Q}^2$. A construction of Ref. [47] shows that this can be improved:

Theorem 52.13 [47]

There is a periodic, plane covering graph G_∞ of detour 1.67784..., such that each finite set of rational points is contained in a finite part of a scaled copy of G_∞.

On the other side of the spectrum Ref. [47] gives an example of a point set P with detour $\dot{\delta}(P) \geq \pi/2 = 1.57079\ldots$. Subsequent work in Refs. [54,55] improved this lower bound by exhibiting a point set P with $\dot{\delta}(P) > \pi/2$.

Theorem 52.14 **[47,55]**

- *Let P be the vertex set of the regular n-gon on the unit circle. Then $\dot{\delta}(P) \geq \pi/2$.*
- *Let $P = \{(x, y) \mid x, y \in \{-9, \ldots, 9\}\}$. Then $\dot{\delta}(P) \geq (1 + 10^{-11})\pi/2$.*

52.5 Low-Dilation Networks

Besides computing the dilation of given graphs, the problem of constructing certain finite plane graphs $G = (V, E)$ of small dilation that contain a given finite point set P is also interesting. There are several different variants of this problem, depending on whether G may or may not contain Steiner-points, or if G is restricted to belong to a certain class of graphs $\mathcal{G}$, like, for example, triangulations and trees.

Definition 52.4 (Dilation of a point set)

Let $\mathcal{G}$ be a class of graphs and P be a finite point set in the plane. The dilation $\dot{\Delta}^{\mathcal{G}}(P)$ *of P wrt. $\mathcal{G}$ is the smallest possible dilation of any finite plane graph $G = (P, E)$ in $\mathcal{G}$, that is,*

$$\dot{\Delta}^{\mathcal{G}}(P) := \min_{\substack{G=(P,E) \in \mathcal{G} \\ G \text{ finite, plane}}} \Delta(G)$$

If $\mathcal{G}$ is the class of all graphs, we can assume G to be a triangulation, since an optimal solution only contains straight edges, and adding edges never increases the dilation. We omit the superscript $\mathcal{G}$ in this case.

52.5.1 Triangulations

A triangulation defining $\dot{\Delta}(P)$ is called a *minimum dilation triangulation* of P. So far, only little research has been conducted on minimum dilation triangulations. The complexity status of the problem is open. Most work upperbounds the dilation of certain types of triangulations. Chew [56] has shown that the rectilinear Delaunay triangulation has dilation at most $\sqrt{10}$. Dobkin et al. [57] give a similar result for the Euclidean Delaunay triangulation. They show that its dilation can be bounded from above by $((1+\sqrt{5})/2)\pi \approx 5.08$. This bound was further improved to $2\pi/\,(3\cos(\pi/6)) \approx 2.42$ by Keil and Gutwin [58,59]. Das and Joseph [60] generalize all these results by identifying two properties of planar graphs such that if A is an algorithm that computes a planar graph from a given set of points and if all the graphs constructed by A meet these properties, then the dilation of all the graphs constructed by A is bounded by a constant.

Exclusion and Inclusion Regions

When investigating optimal triangulations, it is usually instructive to consider local properties of the edges in these triangulations. One important class of local properties that has been studied extensively, for example, for minimum-weight triangulations are *exclusion regions.* They provide a necessary condition for the inclusion of an edge into an optimal triangulation: If p and q are two points in P, then the edge $e := \overline{pq}$ can only be contained in an optimal triangulation of P if no other points of P lie in (certain parts of) the exclusion region of e.

To obtain an exclusion region for the minimum dilation triangulation one can observe the following: We know from Ref. [58] that the dilation of the Delaunay triangulation of P is bounded by $\gamma = 2\pi/(3\cos(\pi/6))$. Moreover, if we have an edge e and two points x, y on opposite sides of e that are close to the center of e, then the dilation between x and y is large, because e constitutes an obstacle that the shortest path between x and y has to surpass. In fact, if we can quantify this, and show that the dilation between any pair of points in a certain region $D_{e,\gamma}$ that lie on opposing sides of e is larger than γ, we can conclude that, if $D_{e,\gamma}$ contains such a pair of points, then e cannot be contained in the minimum dilation triangulation of P, since the Delaunay triangulation gives a better dilation than any triangulation containing e.

The upper bound of Ref. [58] can also be used to obtain a *sufficient* condition for the inclusion of an edge. More specifically, consider for two points $p, q \in P$ the ellipsoid $E_{p,q,\gamma}$ with foci p and q that is given by $E_{p,q,\gamma} = \{x \in \mathbb{R}^2 || px| + |qx| \leq \gamma \cdot |pq|\}$. If $E_{p,q,\gamma}$ is empty, then the line segment $\overline{pq}$ has to be included in the minimum dilation triangulation of P, since otherwise the dilation between p and q would be larger than γ.

Theorem 52.15 [61,62]

Let $\gamma = 2\pi/(3\cos(\pi/6))$ *and* $p, q \in P$.

1. *For any* $0 < \alpha < 1/(2\gamma)$ *the disc* $D_{\overline{pq},\alpha}$ *of radius* $\alpha|pq|$ *centered at the midpoint of* $\overline{pq}$ *is an exclusion region for the minimum dilation triangulation.*
2. *The ellipsoid* $E_{p,q,\gamma} = \{x \in \mathbb{R}^2 || px| + |qx| \leq \gamma \cdot |pq|\}$ *is an inclusion region for the minimum dilation triangulation.*

Regular n-Gons

Even for the vertex set $S_n = \{s_1, \ldots, s_n\}$ of a regular n-gon, it is not known how to efficiently compute a minimum dilation triangulation. There is however some additional understanding of the structure of optimal triangulations in that case. In particular, there is a simple lower bound for the dilation of S_n.

Theorem 52.16 [62]

Let $n \geq 74$, *and assume that* $\max_{1 \leq i < j \leq n} ||s_i - s_j|| = 2$. *For any triangulation* T *of* S_n *and any maximum dilation pair* $s_x, s_y \in S_n$ *of* T, *we have*

1. $\Delta(T) \geq \sqrt{2 - \sqrt{3}} + \sqrt{3}/2 \approx 1.3836$,
2. $|x - y| > 5n/12$, *and*
3. $||s_x - s_y|| > \left(\sqrt{6 + 3\sqrt{3}} + \sqrt{2 - \sqrt{3}}\right)/2 \approx 1.93185$.

This can be used to derive an efficient approximation algorithm that computes a triangulation whose dilation is within a factor of $1 + O(1/\sqrt{\log n})$ of the optimum.

Theorem 52.17 [62]

In $O(n\sqrt{\log n})$ *time a triangulation* T^* *of* S_n *can be computed, such that*

$$\Delta(T^*) \leq \left(1 + O\left(1/\sqrt{\log n}\right)\right) \dot{\Delta}(S_n)$$

52.5.2 Stars

The problem of computing a minimum dilation star of P, that is, a graph $G \in \mathcal{G}$ defining $\dot{\Delta}^P$ where $\mathcal{G}$ is a star, was considered for the first time by Eppstein and Wortman [36]. They proved the following theorem.

Theorem 52.18

Let P *be a set of* n *points in* $\mathbb{E}^2$; *one can compute a minimum dilation star of* P *in* $O(n2^{\alpha(n)} \log n)$ *expected time, where* α *is the functional inverse of Ackermann's function [63].*

The algorithm works by iteratively selecting a random vertex c in a region R, evaluating the dilation that would result from using c as a center, computing the region R that could contain a center yielding a lower dilation, and discarding the vertices outside R. Evaluating the dilation Δ_c of a given star with center at c in $O(n \log n)$ time was discussed in Section 52.3.

The region R is the intersection of $O(n)$ ellipses defined by the $O(n)$ pairs of points identified in Section 52.3, that is, for each of the pairs v_i and v_j the level set $f_{i,j}^{\leq\lambda} = \{x \in \mathbb{E}^2 | f_{i,j}(x) = \frac{|v_i x| + |x v_j|}{|v_i v_j|} \leq \lambda\}$

defines an ellipsoid with foci v_i and v_j. The intersection of those $O(n)$ ellipses can be described by $O(n2^{\alpha(n)})$ arcs.

In each iteration, any vertex in R will be removed with probability 1/2; so the expected number of iterations is $O(\log n)$, resulting in an $O(n2^{\alpha(n)} \log n)$ expected time algorithm.

52.5.3 Small Spanners with Small Dilation

Aronov et al. [64] considered the problem of constructing a minimum dilation graph given the number of edges as a parameter. Any spanner of a set of n points S must have at least $n-1$ edges, because otherwise the graph would not be connected and the dilation would be infinite. The quantity $\Delta(S, k)$ is defined as

$$\Delta(S, k) = \min_{\substack{V(G)=S \\ |E(G)| = n-1+k}} \Delta(G)$$

Thus, $\Delta(S, k)$ is the minimum dilation one can achieve with a network on S that has $n-1+k$ edges.

Theorem 52.19

For any n and any k with $1 \leq k \leq 2n$, there is a set S of n points such that any graph on S with $n-1+k$ edges has dilation at least $\frac{2}{\pi} \cdot \lfloor \frac{n}{k+1} \rfloor - 1$.

Consider a set S of n points $p_1, \ldots, p_n$ spaced equally on the unit circle, and let o be the center of the circle. The first step is to prove a lower bound on any tree T for S.

Let x and y be two points in S and let γ and γ' be two paths from x to y avoiding o. The paths γ and γ' are *(homotopy) equivalent* if γ and γ' belong to the same homotopy class in the punctured plane $\mathbb{R}^2 \setminus \{o\}$. Let γ_i be the unique path in T from p_i to p_{i+1} (where $p_{n+1} := p_1$). Aronov et al. [64] prove that there must be at least one index i for which γ_i is not equivalent to the straight segment $p_i p_{i+1}$. Since the path γ_i must not only "go around" o, but must do so using points p_j on the circle only it follows that $\Delta(S, 0) \geq \frac{2}{\pi} n - 1$.

Theorem 52.19 follows from the above argument by letting S consist of $k+1$ copies of the above construction, that is, sets S_i, for $1 \leq i \leq k+1$, each consisting of at least $\lfloor n/(k+1) \rfloor$ points. The points in S_i are placed equally spaced on a unit–radius circle with center at $(2 \cdot i \cdot n, 0)$. The set S is the union of $S_1, \ldots, S_{k+1}$.

In the same paper, Aronov et al. [64] give a matching upper bound. The algorithm constructs a graph $G = (S, E)$ with $n-1+k$ edges, and dilation $O(n/(k+1))$ in $O(n \log n)$ time.

Let $m \leftarrow \lfloor (k+5)/2 \rfloor$. Partition a minimum spanning tree T of S into m disjoint connected subtrees, $T_1, \ldots, T_m$, each containing $O(n/m)$ points. The edges of each subtree is added to E. Next, consider a Delaunay triangulation of S. For each pair of subtrees T_i and T_j the shortest Delaunay edge (if any) is added to E. This completes the construction of $G = (S, E)$.

The number of edges in G is at most $n-1+k$ and $\Delta(G)$ can be bounded by $O(n/(k+1))$ [64].

Acknowledgments

We would like to thank Michiel Smid for valuable comments on an earlier draft of this chapter.

References

[1] Arya, S., Das, G., Mount, D. M., Salowe, J. S., and Smid, M., Euclidean spanners: short, thin, and lanky, *Proc. of STOC*, 1995, p. 489.

[2] Agarwal, P. K., Wang, Y., and Yin, P., Lower bound for sparse Euclidean spanners, *Proc. SODA*, 2005, p. 670.

[3] Chen, D. Z., Das, G., and Smid, M., Lower bounds for computing geometric spanners and approximate shortest paths, *Disc. Appl. Math.*, 110, 151, 2001.

[4] Narasimhan, G. and Smid, M., *Geometric Spanner Networks*, Kluwer, Dordrecht, 2006.

[5] Clarkson, K. L., Approximation algorithms for shortest path motion planning, *Proc. ACM Symp. on Computational Geom.*, 1987, p. 56.

[6] Keil, J. M., Approximating the complete Euclidean graph, *Proc. Scandinavian Workshop on Algorithmic Theory*, Lecture Notes in Computer Science, Vol. 382, Springer, Berlin, 1988, p. 208.

[7] Althöfer, I., Das, G., Dobkin, D. P., Joseph, D., and Soares, J., On sparse spanners of weighted graphs, *Disc. Comput. Geom.*, 9(1), 81, 1993.

[8] Ruppert, J. and Seidel, R., Approximating the d-dimensional complete Euclidean graph, *Proc. Canadian Conf. on Computational Geom.*, 1991, p. 207.

[9] Yao, A. C., On constructing minimum spanning trees in k-dimensional spaces and related problems, *SIAM J. Comput.*, 11(4), 721–736, 1982.

[10] Pugh, W., Skip lists: a probabilistic alternative to balanced trees, *CACM*, 33(6), 668, 1990.

[11] Bose, P., Gudmundsson, J., and Morin, P., Ordered theta graphs, *Comput. Geom.—Theor. Appl.*, 28, 11, 2004.

[12] Chandra, B., Das, G., Narasimhan, G., and Soares, J., New sparseness results on graph spanners, *Int. J. Comput. Geom. Appl.*, 5, 124, 1995.

[13] Lenhof, H.-P., Salowe, J. S., and Wrege, D. E., New methods to mix shortest-path and minimum spanning trees, Unpublished manuscript, 1993.

[14] Arya, S. and Smid, M., Efficient construction of a bounded-degree spanner with low weight, *Algorithmica*, 17, 33, 1997.

[15] Callahan, P. B. and Kosaraju, S. R., A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields, *JACM*, 42, 67, 1995.

[16] Callahan, P. B. and Kosaraju, S. R., Faster algorithms for some geometric graph problems in higher dimensions, *Proc. SODA*, 1993, p. 291.

[17] Salowe, J. S., Constructing multidimensional spanner graphs, *Int. J. Comput. Geom. Appl.*, 1(2), 99, 1991.

[18] Salowe, J. S., Enumerating interdistances in space, *Int. J. Comput. Geom. Appl.*, 2(1), 49, 1992.

[19] Vaidya, P. M., Minimum spanning trees in k-dimensional space, *SIAM J. Comput.*, 17, 572, 1988.

[20] Vaidya, P. M., An $O(n \log n)$ algorithm for the all-nearest-neighbors problem, *Disc. Comput. Geom.*, 4, 101, 1989.

[21] Vaidya, P. M., A sparse graph almost as good as the complete graph on points in K dimensions, *Disc. Comput. Geom.*, 6, 369, 1991.

[22] Arya, S., Mount, D. M., and Smid, M., Randomized and deterministic algorithms for geometric spanners of small diameter, *Proc. FOCS*, 1994, p. 703.

[23] Chandra, B., Constructing sparse spanners for most graphs in higher dimensions, *Inf. Proc. Lett.*, 51(6), 289, 1994.

[24] Das, G., Heffernan, P., and Narasimhan, G., Optimally sparse spanners in 3-dimensional Euclidean space, *Proc. Symp. on Computational Geom.*, 1993, p. 53.

[25] Das, G. and Narasimhan, G., A fast algorithm for constructing sparse Euclidean spanners, *Int. J. Comput. Geom. Appl.*, 7, 297, 1997.

[26] Das, G., Narasimhan, G., and Salowe, J., A new way to weigh malnourished Euclidean graphs, *Proc. of SODA*, 1995, p. 215.

[27] Gudmundsson, J., Levcopoulos, C., and Narasimhan, G., Improved greedy algorithms for constructing sparse geometric spanners, *SIAM J. Comput.*, 31(5), 1479, 2002.

[28] Soares, J., Approximating Euclidean distances by small degree graphs, *Disc. Comput. Geom.*, 11, 213, 1994.

[29] Mount, D., Personal communication, 1998.

[30] Navarro, G. and Paredes, R., Practical construction of metric t-spanners, *Proc. Workshop on Algorithm Engineering and Experiments*, 2003, p. 69.

[31] Sigurd, M. and Zachariasen, M., Construction of minimum-weight spanners, *Proc. European Symp. on Algorithms*, Lecture Notes in Computer Science, Vol. 3221, Springer, Berlin, 2004, p. 797.

[32] Farshi, M. and Gudmundsson, J., Experimental study of geometric t-spanners, *Proc. 13th European Symp. on Algorithms*, Lecture Notes in Computer Science, Springer, Berlin, 2005.

[33] Langerman, S., Morin, P., and Soss, M., Computing the maximum detour and spanning ratio of planar chains, trees and cycles, *Proc. STACS*, Lecture Notes in Computer Science, Vol. 2285, Springer, Berlin, 2002, p. 250.

[34] Agarwal, P. K., Klein, R., Knauer, C., and Sharir, M., Computing the Detour of Polygonal Curves, TR B 02-03, Freie Universität Berlin, Fachbereich Mathematik und Informatik, 2002.

[35] Agarwal, P. K., Klein, R., Knauer, C., Langerman, S., Morin, P., Sharir, M., and Soss, M., Computing the detour and spanning ratio of paths, trees and cycles in 2D and 3D, *Disc. Comput. Geom.*, Accepted for publication.

[36] Eppstein, D. and Wortman, K. A., Minimum dilation stars, *Proc. Symp. on Computational Geom.*, 2005, p. 321.

[37] Narasimhan, G. and Smid, M., Approximating the stretch factor of Euclidean graphs, *SIAM J. Comput.*, 30(3), 978, 2000.

[38] Arikati, S. R., Chen, D. Z., Chew, L. P., Das, G., Smid, M., and Zaroliagis, C. D., Planar spanners and approximate shortest path queries among obstacles in the plane, *Proc. European Symp. on Algorithms*, Lecture Notes in Computer Science, Vol. 1136, Springer, Berlin, 1996, p. 514.

[39] Gudmundsson, J., Levcopoulos, C., Narasimhan, G., and Smid, M., Approximate distance oracles for geometric graphs, *Proc. of SODA*, 2002, p. 828.

[40] Gudmundsson, J., Levcopoulos, C., Narasimhan, G., and Smid, M., Approximate distance oracles revisited, *Proc. Intl. Symp. on Algorithms and Computation*, Lecture Notes in Computer Science, Vol. 2518, Springer, Berlin, 9, 2002, p. 357.

[41] Cohen, E., Fast algorithms for constructing t-spanners and paths with stretch t, *SIAM J. Comput.*, 28, 210, 1998.

[42] Icking, C. and Klein, R., Searching for the kernel of a polygon: a competitive strategy, *Proc. ACM Symp. Comput. Geom.*, 1995, p. 258.

[43] Alt, H. and Guibas, L. J., Discrete geometric shapes: matching, interpolation, and approximation, in *Handbook of Computational Geometry*, Sack, J. R. and Urrutia, J., Eds., Elsevier Science Publishers, Amsterdam, 2000, p. 121.

[44] Alt, H., Knauer, C., and Wenk, C., Bounding the Fréchet distance by the Hausdorff distance, in *Abstracts European Workshop Comput. Geom.*, Freie Universität Berlin, 2001, p. 166.

[45] Alt, H., Knauer, C., and Wenk, C., Comparison of distance measures for planar curves, *Algorithmica*, 38(2), 45, 2004.

[46] Ebbers-Baumann, A., Klein, R., Langetepe, E., and Lingas, A., A fast algorithm for approximating the detour of a polygonal chain, *Comput. Geom.—Theor. and Appl.*, 27, 123, 2004.

[47] Ebbers-Baumann, A., Grüne, A., and Klein, R., On the geometric dilation of finite point sets, *Proc. Int. Symp. on Algorithms and Computation*, Lecture Notes in Computer Science, Vol. 2906, Springer, Berlin, 2003, p. 250.

[48] Grüne, A., Klein, R., and Langetepe, E., Computing the detour of polygons, *Abstracts of the European Workshop on Computational Geom.*, 2003, p. 61.

[49] Grüne, A., Umwege in Polygonen, Master thesis, Universität Bonn, Germany, 2002.

[50] Erickson, J., New lower bounds for Hopcroft's problem, *Disc. Comput. Geom.*, 16, 389, 1996.

[51] Henzinger, M., Klein, P. N., Rao, S., and Subramanian, S., Faster shortest-path algorithms for planar graphs, *JCSS*, 55(1), 3, 1997.

[52] Chan, T. M., Geometric applications of a randomized optimization technique, *Disc. Comput. Geom.*, 22(4), 547, 1999.

[53] Megiddo, N., Applying parallel computation algorithms in the design of serial algorithms, *JACM*, 30(4), 852, 1983.

[54] Ebbers-Baumann, A., Grüne, A., and Klein, R., Geometric dilation of closed planar curves: a new lower bound, *Abstracts of the European Workshop on Computational Geom.*, 2004, p. 123.

[55] Dumitrescu, A., Grüne, A., and Rote, G., Improved lower bound on the geometric dilation of point sets, *Abstracts of the European Workshop on Computational Geom*, 2005, p. 37.

[56] Chew, L. P., There are planar graphs almost as good as the complete graph, *JCSS*, 39, 205, 1989.

[57] Dobkin, D. P., Friedman, S. J., and Supowit, K. J., Delaunay graphs are almost as good as complete graphs, *Disc. Comput. Geom.*, 5, 399, 1990.

[58] Keil, J. M. and Gutwin, C. A., The Delaunay triangulation closely approximates the complete Euclidean graph, *Proc. Workshop Algorithms Data Struct.*, Lecture Notes in Computer Science, Vol. 382, Springer, Berlin, 1989, p. 47.

[59] Keil, J. M. and Gutwin, C. A., Classes of graphs which approximate the complete Euclidean graph, *Disc. Comput. Geom.*, 7, 13, 1992.

[60] Das, G. and Joseph, D., Which triangulations approximate the complete graph? *Proc. Intl. Symp. on Optimal Algorithms*, Lecture Notes in Computer Science, Vol. 401, Springer, Berlin, 1989, p. 168.

[61] Knauer, C. and Mulzer, W., An exclusion region for minimum dilation triangulations, *Proc. European Workshop on Computational Geom.*, 2005, p. 33.

[62] Knauer, C. and Mulzer, W., Minimum Dilation Triangulations, TR B 05-06, Freie Universität Berlin, Fachbereich Mathematik und Informatik, 2005.

[63] Ackermann, W., Zum Hilbertschen Aufbau der reellen Zahlen, *Math. Ann.*, 99, 118, 1928.

[64] Aronov, B., de Berg, M., Cheong, O., Gudmundsson, J., Haverkort, H., and Vigneron, A., Sparse geometric graphs with small dilation, *Proc. Intl. Symp. on Algorithms and Computation,* Lecture Notes in Computer Science, Vol. 3827, Springer, Berlin, 2005, pp. 50–59.

53

The Well-Separated Pair Decomposition and Its Applications

Michiel Smid
Carleton University

53.1 Introduction

Computational geometry is concerned with the design and analysis of algorithms that solve problems on geometric data in $\mathbb{R}^d$, where the dimension d is considered to be a constant. A large part of the field has been devoted to problems that involve distances determined by pairs of points in a given point set. Given a set S of n points in $\mathbb{R}^d$, we may wish to compute a pair p, q of distinct points in S whose distance is minimum, the k smallest distances among the $\binom{n}{2}$ pairwise distances, the nearest neighbor of each point of S, or the minimum spanning tree of S. Most problems of this type can be rephrased as a graph problem on the complete Euclidean graph on S, in which each edge (p, q) has a weight being the Euclidean distance $|pq|$ between p and q. Since the number of edges in this graph is $\Theta(n^2)$, many problems involving pairwise distances can trivially be solved in $O(n^2)$ time. Even though the complete Euclidean graph has size $\Theta(n^2)$, it can be represented in $\Theta(n)$ space: It is clearly sufficient to only store the points of S, because the weight of any edge can be computed in $O(1)$ time. This leads to the question whether distance problems can be solved in subquadratic time, possibly at the cost of obtaining an approximate solution. For many of these problems, subquadratic algorithms have indeed been designed; see, for example, the books by Preparata and Shamos [1] and de Berg et al. [2], and the survey papers by Bern and Eppstein [3], Eppstein [4], and Smid [5]. Most of these algorithms, however, are tailored to the problem at hand.

Callahan and Kosaraju [6,7] devised the *well-separated pair decomposition (WSPD)* and showed that it can be used to solve a large variety of distance problems. Intuitively, a WSPD is a partition of the $\binom{n}{2}$ edges of the complete Euclidean graph into $O(n)$ subsets. Each subset in this partition is represented by two subsets A and B of the point set S, such that (i) all distances between points in A and points in B are approximately equal and (ii) all distances between points in A (resp. B) are much smaller than distances between A and B. Thus, a WSPD can be regarded as a set of $O(n)$ edges that approximates the dense complete Euclidean graph.

Callahan and Kosaraju [6,7] showed that the WSPD can be used to obtain optimal algorithms for solving the closest pair problem, the k closest pairs problem, the all-nearest neighbors problem, and the approximate minimum spanning tree problem. In recent years, other researchers have shown that the WSPD can be used to solve many other problems. In this chapter, we give an overview of several proximity problems that can be solved efficiently using the WSPD. We mention that the WSPD has also been a critical tool for the solution of several variants of the problem of constructing spanners; an overview of these results can be found in Chapter 52 by Gudmundsson and Knauer in this handbook. An extensive treatment of the WSPD and its applications is given in the book by Narasimhan and Smid [8].

The rest of this chapter is organized as follows. In Section 53.2, we define the WSPD. In Section 53.3, we present an efficient algorithm for constructing a WSPD. In Section 53.4, we show that the WSPD can be used to obtain optimal algorithms for the closest pair problem, the k closest pairs problem, and the all-nearest neighbors problem. In Section 53.5, we use the WSPD to obtain approximate solutions for the diameter problem, the spanner problem, the minimum spanning tree problem, and the problem of computing the kth closest pair. Finally, in Section 53.6, we mention recent work in which the WSPD has been generalized to more general metric spaces.

53.2 Well-Separated Pairs

In this section, we define the WSPD and prove one of its main properties in Lemma 53.1. As mentioned in Section 53.1, the WSPD was introduced by Callahan and Kosaraju [6,7]. Previously, however, similar ideas were used by Salowe [9,10] and Vaidya [11–13], who designed efficient algorithms for computing spanners, all-nearest neighbors, and k closest pairs.

We start by defining the notion of two sets being well-separated. For any set X of points in $\mathbb{R}^d$, we denote its *bounding box* by $R(X)$. Thus, $R(X)$ is the smallest axes-parallel hyperrectangle that contains the set X.

Definition 53.1

Let A and B be two finite sets of points in $\mathbb{R}^d$ and let $s > 0$ be a real number. We say that A and B are well-separated *with respect to s, if there exists two disjoint balls C_A and C_B, such that*

1. *C_A and C_B have the same radius,*
2. *C_A contains $R(A)$,*
3. *C_B contains $R(B)$, and*
4. *the distance between C_A and C_B is at least s times the radius of C_A.*

The real number s is called the separation ratio.

If we are given the bounding boxes $R(A)$ and $R(B)$ of the sets A and B, respectively, then we can test in $O(1)$ time whether these two sets are well-separated.

In the next lemma, we prove the two properties of well-separated sets that were mentioned already in Section 53.1: If A and B are well-separated with respect to a large separation ratio s, then (i) all distances between points in A and points in B are approximately equal and (ii) all distances between points in A (resp. B) are much smaller than distances between A and B. These two properties will be used repeatedly in the rest of this chapter.

Lemma 53.1

Let $s > 0$ be a real number, let A and B be two sets in $\mathbb{R}^d$ that are well-separated with respect to s, let a and a' be two points in A, and let b and b' be two points in B. Then, we have

1. $|aa'| \leq (2/s)|ab|$, *and*
2. $|a'b'| \leq (1 + 4/s)|ab|$.

Proof

Let C_A and C_B be disjoint balls of the same radius, say ρ, such that $R(A) \subseteq C_A$, $R(B) \subseteq C_B$, and the distance between C_A and C_B is at least $s\rho$. The first claim follows from the facts that $|aa'| \leq 2\rho$ and $|ab| \geq s\rho$. By combining the first claim with the triangle inequality, we obtain

$$|a'b'| \leq |a'a| + |ab| + |bb'| \leq (1 + 4/s)|ab|$$

proving the second claim. □

Definition 53.2

Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A well-separated pair decomposition (WSPD) *for S, with respect to s, is a sequence*

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_m, B_m\}$$

of pairs of nonempty subsets of S, for some integer m, such that

1. *for each i with $1 \leq i \leq m$, A_i and B_i are well-separated with respect to s, and*
2. *for any two distinct points p and q of S, there is exactly one index i with $1 \leq i \leq m$, such that*
   (a) *$p \in A_i$ and $q \in B_i$, or*
   (b) *$p \in B_i$ and $q \in A_i$.*

The integer m is called the size *of the WSPD.*

Observe that a WSPD always exists: If we let any two distinct points p and q of S form a pair $\{\{p\}, \{q\}\}$, then the conditions in Definition 53.2 are satisfied. The size of this WSPD, however, is $\binom{n}{2}$. In the next section, we will give an algorithm that computes a WSPD whose size is only $O(n)$.

53.3 Computing a Well-Separated Pair Decomposition

Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ denote the separation ratio. The algorithm that constructs a WSPD for S consists of two phases. In the first phase, a so-called split tree is constructed, which can be considered to be a hierarchical decomposition of the bounding box of S into axes-parallel hyperrectangles. In the second phase, the split tree is used to actually compute the WSPD. The algorithm is due to Callahan and Kosaraju [7].

53.3.1 The Split Tree

The *split tree* $T(S)$ for the point set S is a binary tree that is defined as follows:

1. If $n = 1$, then $T(S)$ consists of a single node storing the only element of S.
2. Assume that $n \geq 2$ and consider the bounding box $R(S) = \Pi_{i=1}^{d}[\ell_i, r_i]$ of S. Let i be the dimension such that $r_i - \ell_i$ is maximum, and define $S_1 := \{p \in S : p_i \leq (\ell_i + r_i)/2\}$ and $S_2 := S \setminus S_1$. The split tree $T(S)$ for S consists of a root whose two children are recursively defined split trees $T(S_1)$ and $T(S_2)$ for the sets S_1 and S_2, respectively. The root of $T(S)$ stores the bounding box $R(S)$.

Thus, the split tree $T(S)$ stores the points of S at its leaves. Each internal node of $T(S)$ stores an axes-parallel hyperrectangle, which is the bounding box of the set of all points of S that are stored in its subtree. Observe that the split tree is, in general, not balanced.

The above definition immediately leads to an $O(n^2)$-time algorithm for constructing the split tree. Callahan and Kosaraju show that, by using a divide-and-conquer approach, the split tree can in fact be constructed in $O(n \log n)$ time:

Lemma 53.2

The split tree for any set of n points in $\mathbb{R}^d$ can be constructed in $O(n \log n)$ time.

53.3.2 Using the Split Tree to Compute Well-Separated Pairs

Consider the split tree $T = T(S)$ for the point set S. For any node u of T, we denote by S_u the subset of S that is stored at the leaves of the subtree rooted at u. For any subset X of $\mathbb{R}^d$, we denote by $L_{\max}(R(X))$ the length of a longest side of the bounding box $R(X)$ of X.

The following algorithm uses the split tree to compute a WSPD for S. Recall that s denotes the separation ratio.

Step 1: Initialize an empty queue Q. For each internal node u of T, do the following: Let v and w be the two children of u. Insert the pair (v, w) into Q.

Step 2: Repeat the following until the queue Q is empty: Take the first pair (v, w) in Q and delete it from Q. If the sets S_v and S_w are well-separated with respect to s, then output the pair $\{S_v, S_w\}$. Otherwise, assume without loss of generality that $L_{\max}(R(S_v)) \leq L_{\max}(R(S_w))$. Let w_1 and w_2 be the two children of w. Insert the pairs (v, w_1) and (v, w_2) into the queue Q.

It is not difficult to see that the output of this algorithm is a WSPD of the point set S. Callahan and Kosaraju use a nontrivial packing argument to show that the number of pairs is $O(n)$.

Lemma 53.3

Given the split tree, the above algorithm constructs, in $O(s^d n)$ time, a WSPD for S that consists of $O(s^d n)$ pairs.

Observe that the WSPD is represented implicitly by the split tree: Each pair $\{A, B\}$ in the WSPD is represented by two nodes v and w, such that $A = S_v$ and $B = S_w$. Thus, the entire WSPD can be represented using $O(s^d n)$ space.

By combining Lemmas 53.2 and 53.3, we obtain the following result:

Theorem 53.1 **(Callahan and Kosaraju** [7])

Given a set S of n points in $\mathbb{R}^d$, and given a real number $s > 0$, a WSPD for S, with separation ratio s, consisting of $O(s^d n)$ pairs, can be computed in $O(n \log n + s^d n)$ time.

In some applications, it is useful to have a WSPD in which each well-separated pair consists of two sets, at least one of which is a singleton set. For the WSPD $\{A_i, B_i\}$, $1 \leq i \leq m$, that is constructed by the algorithm given above, Callahan [6] proves that

$$\sum_{i=1}^{m} \min(|A_i|, |B_i|) = O(s^d n \log n)$$

Thus, if we replace each pair $\{A_i, B_i\}$ (where we assume without loss of generality that $|A_i| \leq |B_i|$) by $|A_i|$ pairs $\{\{a\}, B_i\}$, $a \in A_i$, then we obtain the following result:

Theorem 53.2 **(Callahan** [6])

Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. In $O(s^d n \log n)$ time, a WSPD for S, with separation ratio s, can be constructed, such that each well-separated pair consists of two sets, at least one of which is a singleton set, and the total number of pairs is $O(s^d n \log n)$.

53.4 Exact Algorithms for Proximity Problems

As we have mentioned before, the WSPD is an $O(n)$-size approximation of the set of $\Theta(n^2)$ distances determined by a set of n points. In this section, we show that, despite the fact that the WSPD *approximates* all distances, it can be used to solve several proximity problems *exactly*. All results in this section are due to Callahan and Kosaraju [6,7].

Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. Consider the split tree T and the corresponding WSPD for S, with separation ratio s, consisting of the pairs $\{A_i, B_i\}$, $1 \leq i \leq m$.

53.4.1 The Closest Pair Problem

In this problem, we want to compute a *closest pair* in S, that is, two distinct points p and q in S such that $|pq|$ is minimum. Many algorithms are known that solve this problem optimally in $O(n \log n)$ time (see Ref. [5]). We show that the WSPD basically "contains" a solution to the closest pair problem.

Let (p, q) be a closest pair in S, and let i be the index such that $p \in A_i$ and $q \in B_i$. If we assume that the separation ratio s is a constant greater than 2, then it follows from Lemma 53.1 that both A_i and B_i are singleton sets. Thus, by considering all pairs $\{A_j, B_j\}$, $1 \leq j \leq m$, such that both A_j and B_j are represented by leaves of the split tree, we obtain the closest pair in $O(m)$ time. Combining this with Theorem 53.1, we obtain the following result:

Theorem 53.3

Given a set S of n points in $\mathbb{R}^d$, a closest pair in S can be computed in $O(n \log n)$ time.

53.4.2 The *k* Closest Pairs Problem

We next consider the problem of computing the *k closest pairs* in S, for any given integer k with $1 \leq k \leq \binom{n}{2}$. That is, we want to compute the k smallest elements in the (multi)set of $\binom{n}{2}$ distances determined by pairs of points in S. Several algorithms have been designed that solve this problem optimally in $O(n \log n + k)$ time (see Ref. [5]). As for the closest pair problem, we show that, once the WSPD is given, the k closest pairs can be obtained in a simple way.

For each i with $1 \leq i \leq m$, we denote by $|R(A_i)R(B_i)|$ the minimum distance between the bounding boxes $R(A_i)$ and $R(B_i)$ of A_i and B_i, respectively. We assume, for ease of notation, that the pairs in the WSPD are numbered such that

$$|R(A_1)R(B_1)| \leq |R(A_2)R(B_2)| \leq \cdots \leq |R(A_m)R(B_m)|$$

(This ordering of the pairs is only used in the analysis, it is *not* computed by the algorithm.) The algorithm does the following:

Step 1: Compute the smallest integer $\ell \geq 1$, such that $\sum_{i=1}^{\ell} |A_i| \cdot |B_i| \geq k$.

Step 2: Compute the distance r between the bounding boxes $R(A_\ell)$ and $R(B_\ell)$ of the sets A_ℓ and B_ℓ, respectively.

Step 3: Compute the largest index ℓ' such that $|R(A_{\ell'})R(B_{\ell'})| \leq (1 + 4/s)r$.

Step 4: Compute the set L' consisting of all pairs (p, q) for which there is an index i with $1 \leq i \leq \ell'$, such that $p \in A_i$ and $q \in B_i$.

Step 5: Return the k smallest distances determined by the pairs in the set L'.

Lemma 53.4

This algorithm computes the k closest pairs in S.

Proof

Let (p, q) be one of the k closest pairs, and let j be the index such that $p \in A_j$ and $q \in B_j$. It suffices to prove that (p, q) is an element of L', that is, $j \leq \ell'$. To prove this, assume that $j > \ell'$. Then

$$|pq| \geq |R(A_j)R(B_j)| > (1 + 4/s)r$$

Let L be the set consisting of all pairs (x, y) for which there is an index i with $1 \leq i \leq \ell$, such that $x \in A_i$ and $y \in B_i$. This set contains at least k elements. Using Lemma 53.1, we have, for each pair

(x, y) in L,

$$|xy| \leq (1+4/s)|R(A_i)R(B_i)| \leq (1+4/s)|R(A_\ell)R(B_\ell)| = (1+4/s)r$$

This contradicts our assumption that (p, q) is one of the k closest pairs. □

Using a linear-time (weighted) selection algorithm, the running time of the algorithm can be bounded by

$$O\left(m + \sum_{i=1}^{\ell'} |A_i| \cdot |B_i|\right) = O(m + |L'|)$$

Let δ be the kth smallest distance in S. Then it can be shown that $r \leq \delta$ and $|pq| \leq (1+4/s)^2\delta$, for any pair (p, q) in L'. Hence, if we denote by M the number of distances in the set S that are at most $(1+4/s)^2\delta$, then the running time of the algorithm is $O(m+M)$. By using a counting technique, based on a grid with cells having sides of length $\delta/\sqrt{d}$, it can be shown that

$$M = O((1+4/s)^{2d}(n+k))$$

We take the separation ratio s to be equal to, say, 1. Then, by combining our results with Theorem 53.1, we obtain the following theorem:

Theorem 53.4

Given a set S of n points in $\mathbb{R}^d$, and given an integer k with $1 \leq k \leq \binom{n}{2}$, the k closest pairs in S can be computed in $O(n \log n + k)$ time.

53.4.3 The All-Nearest Neighbors Problem

In this problem, we want to compute for each point p of S a *nearest neighbor* in S, that is, a point $q \in S\setminus\{p\}$ for which $|pq|$ is minimum. Vaidya [12] was the first to solve this problem optimally in $O(n \log n)$ time. In fact, his algorithm uses ideas that are very similar to the WSPD. In this section, we sketch the algorithm of Callahan and Kosaraju [7].

Let p be a point of S and let q be its nearest neighbor. Let i be the index such that $p \in A_i$ and $q \in B_i$. It follows from Lemma 53.1 that the set A_i consists only of the point p. Hence, to solve the all-nearest neighbors problem, we only have to consider pairs of the WSPD, for which at least one of their sets is a singleton set. This observation does not lead to an efficient algorithm yet.

For any node u of the split tree T, we define $F(u)$ to be the set of all points $p \in S$ such that $\{\{p\}, S_v\}$ is a pair in the WSPD, for some ancestor v of u. (We consider u to be an ancestor of itself.) Also, we define $N(u)$ to be the set of all points $p \in F(u)$, such that the distance from p to the smallest ball containing $R(S_u)$ is at most equal to the smallest distance between p and any other point of $F(u)$. Observe that $N(u) \subseteq F(u)$.

Lemma 53.5

The size of the set $N(u)$ is $O((s/(s-1))^d)$.

Proof

Let C be the smallest ball that contains $R(S_u)$. We claim that

1. for each $p \in N(u)$, the sets $\{p\}$ and S_u are well separated, and
2. $|pC| \leq |pq|$, for any two distinct points p and q of $N(u)$.

By combining these two claims with a generalization of the fact that any point can be the nearest neighbor of at most a constant number of other points, it can be shown that the size of $N(u)$ is $O((s/(s-1))^d)$.

To prove the first claim, let $p \in N(u)$. Since $p \in F(u)$, there is an ancestor v of u such that the sets $\{p\}$ and S_v are well separated. Since S_u is a subset of S_v, the sets $\{p\}$ and S_u are well separated as well.

To prove the second claim, let p and q be two distinct points of $N(u)$. The definition of $N(u)$ implies that the distance between p and C is at most the smallest distance between p and any other point of $F(u)$. In particular, since $q \in F(u)$, we have $|pC| \leq |pq|$. □

We assume from now on that the separation ratio s is a constant greater than 2. The sets $N(u)$, where u ranges over all nodes of the split tree T, can be computed in a top-down fashion, in $O(n)$ total time. How do we use these sets? Let p be any point of S, q a nearest neighbor of p, and u the leaf of the split tree that stores q. Let i be the index such that $A_i = \{p\}$ and $q \in B_i$, and let v be the ancestor of u such that $B_i = S_v$. Then, $p \in F(u)$. Moreover, since $S_u = \{q\}$, the distance between p and the smallest ball containing $R(S_u)$ is equal to $|pq|$, which is at most the distance between p and any other point of $F(u)$. Therefore, we have $p \in N(u)$.

The discussion above, together with Theorem 53.1, leads to an algorithm that solves the all-nearest neighbors problem in $O(n \log n)$ time.

Theorem 53.5

Given a set S of n points in $\mathbb{R}^d$, the all-nearest neighbors problem can be solved in $O(n \log n)$ time.

53.5 Approximation Algorithms for Proximity Problems

In this section, we consider proximity problems for which no optimal exact algorithms are known. For each of these problems, we show that the WSPD leads to simple and fast approximation algorithms.

Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. We assume that we have already computed the split tree T and the corresponding WSPD for S, with separation ratio s, consisting of the pairs $\{A_i, B_i\}$, $1 \leq i \leq m$. For each i with $1 \leq i \leq m$, we choose an arbitrary element a_i in A_i and an arbitrary element b_i in B_i.

53.5.1 The Diameter Problem

The *diameter* of S is defined to be the largest distance between any two points of S. If the dimension d is equal to two, the diameter can be computed in $O(n \log n)$ time (see Ref. [1]). Ramos [14] obtained the same time bound for the three-dimensional case. It is not known if for dimensions greater than three, the diameter can be computed in $O(n \log^{O(1)} n)$ time. In this section, we show that the WSPD leads to a simple and efficient algorithm that approximates the diameter, for any constant dimension d.

Let D be the diameter of S, and let i be the index for which $|a_i b_i|$ is maximum. A straightforward application of Lemma 53.1 shows that $D/(1 + 4/s) \leq |a_i b_i| \leq D$. Hence, if we choose $s = 4(1 - \epsilon)/\epsilon$, then this result, together with Theorem 53.1, yields the following theorem:

Theorem 53.6

Given a set S of n points in $\mathbb{R}^d$, and given a real constant $0 < \epsilon < 1$, a $(1 - \epsilon)$-approximation to the diameter of S can be computed in $O(n \log n)$ time.

53.5.2 The Spanner Problem

For a real number $t > 1$, a graph $G = (S, E)$ is called a *t-spanner* for the point set S, if for any two points p and q of S, we have $|pq|_G \leq t|pq|$, where $|pq|_G$ denotes the length of a shortest path in G between p and q. Many algorithms are known that compute spanners (see Chapter 52 by Gudmundsson and Knauer in this handbook and Ref. [8]). In this section, we show that the WSPD immediately gives a spanner for S consisting of $O(n)$ edges. The construction is due to Callahan and Kosaraju [15].

Lemma 53.6

Assume that the separation ratio s is greater than 4. Define $G = (S, E)$ to be the graph with edge set $E = \{(a_i, b_i) : 1 \leq i \leq m\}$. Then, G is a t-spanner for S, where $t = (s + 4)/(s - 4)$.

Proof

The proof is by induction. Consider two points p and q in S. If $p = q$, then obviously $|pq|_G \leq t|pq|$. Assume that $p \neq q$. Moreover, assume that $|xy|_G \leq t|xy|$, for all points x and y in S for which $|xy| < |pq|$. Let i be the index such that $p \in A_i$ and $q \in B_i$. Using Lemma 53.1, we obtain

$$\begin{aligned} |pq|_G &\leq |pa_i|_G + |a_i b_i|_G + |b_i q|_G \\ &= |pa_i|_G + |a_i b_i| + |b_i q|_G \\ &\leq t|pa_i| + |a_i b_i| + t|b_i q| \\ &\leq (2t/s + (1 + 4/s) + 2t/s)|pq| \\ &= t|pq| \end{aligned}$$

□

Thus, Theorem 53.1 implies the following result:

Theorem 53.7

Given a set S of n points in $\mathbb{R}^d$, and given a real constant $t > 1$, a t-spanner for S, consisting of $O(n)$ edges, can be computed in $O(n \log n)$ time.

53.5.3 The Minimum Spanning Tree Problem

In the two-dimensional case, the minimum spanning tree of S can be computed in $O(n \log n)$ time, by using the fact that it is contained in the Delaunay triangulation of S (see Refs. [1,2]). For the three-dimensional case, the best known algorithm has a running time that is close to $O(n^{4/3})$ (see Ref. [16]). Erickson [17] argues that it is unlikely that this running time can be improved considerably. In this section, we show that, in any constant dimension d, an approximation to the minimum spanning tree can be computed in $O(n \log n)$ time. The algorithm is due to Callahan and Kosaraju [15].

Assume that the separation ratio s is a constant greater than 4, and let $t = (s + 4)/(s - 4)$. Consider again the graph G of Lemma 53.6. Since G is a t-spanner for S, this graph is connected. Let T be a minimum spanning tree of G.

Lemma 53.7

T is a t-approximate minimum spanning tree of S.

Proof

Let T^* be a minimum spanning tree of S, and denote its total edge length by $|T^*|$. Number the edges of T^* as $e_1, e_2, \ldots, e_{n-1}$. For each i with $1 \leq i \leq n - 1$, let P_i be a t-spanner path (in G) between the endpoints of e_i, and denote the length of this path by $|P_i|$. Then,

$$\sum_{i=1}^{n-1} |P_i| \leq \sum_{i=1}^{n-1} t|e_i| = t|T^*|$$

Let G' be the subgraph of G consisting of the union of the edges of all paths P_i, $1 \leq i \leq n - 1$. Then G' is a connected graph on the points of S, and its weight is at most $t|T^*|$. Since the weight of T is at most that of G', the weight of T is at most t times the weight of T^*. □

Since the graph G contains $O(n)$ edges, its minimum spanning tree T can be computed in $O(n \log n)$ time. By combining this with Theorem 53.1, we obtain the following result:

Theorem 53.8

Given a set S of n points in $\mathbb{R}^d$, and given a real constant $\epsilon > 0$, a $(1 + \epsilon)$-approximation to the minimum spanning tree of S can be computed in $O(n \log n)$ time.

53.5.4 The *k*th Closest Pair Problem

In this problem, we are given an integer k with $1 \leq k \leq \binom{n}{2}$, and want to compute the kth smallest element in the (multi)set of $\binom{n}{2}$ distances determined by pairs of points in S. In the two-dimensional case, the best known algorithm for this problem has a running time that is close to $O(n^{4/3})$ (see Ref. [18]). Again, Erickson [17] argues that it is unlikely that a significantly faster algorithm exists. We show that, for any constant dimension, there is a simple and efficient algorithm that approximates the kth closest pair. The results in this section are due to Bespamyatnikh and Segal [19].

Let k be any integer with $1 \leq k \leq \binom{n}{2}$. As in Section 53.4.2, we assume that the pairs in the WSPD are numbered such that

$$|R(A_1)R(B_1)| \leq |R(A_2)R(B_2)| \leq \cdots \leq |R(A_m)R(B_m)|$$

Let $\ell \geq 1$ be the smallest integer such that $\sum_{i=1}^{\ell} |A_i| \cdot |B_i| \geq k$, x an arbitrary element of A_ℓ, and y an arbitrary element of B_ℓ.

Lemma 53.8

If δ is the kth smallest distance in the set S, then $\delta/(1 + 4/s) \leq |xy| \leq (1 + 4/s)\delta$.

Proof

As mentioned in Section 53.4.2, it can be shown that $|R(A_\ell)R(B_\ell)| \leq \delta$. If we combine this fact with Lemma 53.1, then we obtain

$$|xy| \leq (1 + 4/s)|R(A_\ell)R(B_\ell)| \leq (1 + 4/s)\delta$$

Let L be the set consisting of all pairs (a, b) for which there is an index i with $1 \leq i \leq \ell$, such that $a \in A_i$ and $b \in B_i$. Let (a, b) be the pair in L for which $|ab|$ is maximum, and let i be the index such that $a \in A_i$ and $b \in B_i$. Observe that $i \leq \ell$. Since L has size at least k, we have $\delta \leq |ab|$. Therefore (again using Lemma 53.1),

$$\delta \leq |ab| \leq (1 + 4/s)|R(A_i)R(B_i)| \leq (1 + 4/s)|R(A_\ell)R(B_\ell)| \leq (1 + 4/s)|xy| \qquad \square$$

Thus, using Theorem 53.1, we obtain the following result:

Theorem 53.9

Let S be a set of n points in $\mathbb{R}^d$, let k be an integer with $1 \leq k \leq \binom{n}{2}$, let $\epsilon > 0$ be a constant, and let δ be the kth smallest distance in S. In $O(n \log n)$ time, a pair (x, y) of points in S can be computed for which $(1 - \epsilon)\delta \leq |xy| \leq (1 + \epsilon)\delta$.

Surprisingly, computing a pair (x, y) of points in S such that $\delta \leq |xy| \leq (1 + \epsilon)\delta$ is more difficult. To compute such a pair, we use the WSPD of Theorem 53.2. Thus, the WSPD consists of pairs $\{A_i, B_i\}$, $1 \leq i \leq m$, where each set A_i is a singleton set, say $A_i = \{a_i\}$, and $m = O(n \log n)$.

For each i with $1 \leq i \leq m$, we define $d_i = \min\{|a_i b| : b \in B_i\}$ and $D_i = \max\{|a_i b| : b \in B_i\}$. Assume that the pairs in the WSPD are numbered such that $d_1 \leq d_2 \leq \cdots \leq d_m$. Let $\ell \geq 1$ be the smallest integer such that $\sum_{i=1}^{\ell} |B_i| \geq k$, and let $D = \max(D_1, D_2, \ldots, D_\ell)$.

Lemma 53.9

If δ is the kth smallest distance in the set S, then $\delta \leq D \leq (1 + 4/s)\delta$.

Proof

Since $\sum_{i=1}^{\ell} |B_i| \geq k$, the pairs $\{A_i, B_i\}$ with $1 \leq i \leq \ell$, define at least k distances. Since D is the largest among these distances, it follows that $\delta \leq D$.

Let $L = \{(a_i, b) : b \in B_i, i \geq \ell\}$. Then d_ℓ is the minimum distance of any element in L. Since the size of L is greater than $\binom{n}{2} - k$, it follows that $\delta \geq d_\ell$. Let i be the index such that $D = D_i$. By Lemma 53.1, we have $D_i \leq (1 + 4/s)d_i$. Therefore, we have

$$D = D_i \leq (1 + 4/s)d_i \leq (1 + 4/s)d_\ell \leq (1 + 4/s)\delta \qquad \square$$

We obtain the value of D (which is an approximation to the kth smallest distance in S), by computing all values d_i and D_i, $1 \leq i \leq m$. That is, for each point a_i, we compute its nearest and furthest neighbors in the set B_i. We will show how to use the split tree to solve this problem for the case when the points are in $\mathbb{R}^2$. The algorithm uses the following result:

Lemma 53.10

Let V be a set of N points in the plane. There exists a data structure that supports the following operations:

1. *For any given query point $q \in \mathbb{R}^2$, report the nearest neighbor of q in V. The query time is $O(\log^2 N)$.*
2. *For any given query point $q \in \mathbb{R}^2$, report the furthest neighbor of q in V. The query time is $O(\log^2 N)$.*
3. *Insert an arbitrary point into the set V. The amortized insertion time is $O(\log^2 N)$.*

Proof

Consider the Voronoi diagram of the set V, which can be constructed in $O(N \log N)$ time. If we store this diagram, together with a point location data structure, then a nearest-neighbor query can be answered in $O(\log N)$ time (see Refs. [1,2]). If we use the furthest-point Voronoi diagram, then we obtain the same result for furthest-neighbor queries. Unfortunately, these Voronoi diagrams cannot be maintained efficiently under insertions of points. Since nearest-neighbor and furthest-neighbor queries are *decomposable*, however, we can use the *logarithmic method* of Bentley and Saxe [20] to obtain the time bounds that are claimed in the lemma. □

We now show how Lemma 53.10 can be used to compute all values of d_i and D_i, $1 \leq i \leq m$. Consider the split tree T. Recall that for any node u, S_u denotes the subset of S that is stored at the leaves in the subtree of u. We store with each node u, a list L_u consisting of all points a_i such that $B_i = S_u$.

The algorithm traverses the split tree T in postorder. During this traversal, the following invariant is maintained: If u is a node that has been traversed, but none of its proper ancestors has been traversed yet, then u stores the data structure DS_u of Lemma 53.10 for the point set S_u.

The postorder traversal of T does the following. Let u be the current node in this traversal. If u is a leaf storing the point, say, p, then we compute $d_i = D_i = |a_i p|$ for each point a_i in L_u, and we build the data structure DS_u of Lemma 53.10 for the singleton set $S_u = \{p\}$. Assume that the current node u is not a leaf. Let v and w be the two children of u. By the invariant, v and w store the data structure DS_v and DS_w of Lemma 53.10 for the sets S_v and S_w, respectively. Assume without loss of generality that $|S_v| \leq |S_w|$. We do the following: First, we discard the data structure DS_v. Then, we insert each element of S_v into DS_w; this results in the data structure DS_u storing all elements of S_u. Finally, for each element a_i of L_u, we use DS_u to find the nearest and furthest neighbors of a_i in the set S_u, and compute the values of d_i and D_i.

We analyze the total time of this algorithm. Since the WSPD contains $O(n \log n)$ pairs, the total number of nearest-neighbor and furthest-neighbor queries is $O(n \log n)$. Consider any fixed point p of S. If p is inserted into a data structure, then it "moves" to a new set whose size is at least twice the size of the previous set containing p. As a result, p is inserted $O(\log n)$ times. Thus, overall, the total number of insertions is $O(n \log n)$. Lemma 53.10 then implies that the running time of the algorithm is $O(n \log^3 n)$. If we combine this with Theorem 53.2, we obtain the following result:

Theorem 53.10

Let S be a set of n points in $\mathbb{R}^d$, let k be an integer with $1 \leq k \leq \binom{n}{2}$, let $\epsilon > 0$ be a constant, and let δ be the kth smallest distance in S. In $O(n \log^3 n)$ time, a pair (x, y) of points in S can be computed for which $\delta \leq |xy| \leq (1 + \epsilon)\delta$.

53.6 Generalization to Metric Spaces

All results in the previous sections are valid for Euclidean spaces $\mathbb{R}^d$, where d is a constant. In recent years, the WSPD (and its applications) has been generalized to more general metric spaces.

Consider an arbitrary metric space (S, δ), where S is a set of n elements and $\delta : S \times S \longrightarrow \mathbb{R}$ the metric defined on S. For any two subsets A and B of S, we denote the minimum distance between any point in A and any point in B by $\delta(A, B)$, and we denote the diameter of A by $D(A)$. If $s > 0$ is a real number, then we say that A and B are well-separated with respect to s, if

$$\delta(A, B) \geq s \cdot \max(D(A), D(B))$$

Using this generalized notion of being well-separated, we define a WSPD for S as in Definition 53.2.

Gao and Zhang [21] considered the problem of constructing a WSPD for the unit-disk graph metric: Let S be a set of n points in $\mathbb{R}^d$. The *unit-disk graph* is defined to be the graph with vertex set S, in which any two distinct points p and q are connected by an edge if and only $|pq| \leq 1$. If we define $\delta(p, q)$ to be the length of a shortest path between p and q in the unit-disk graph, then (S, δ) is a metric space. Observe that even though the unit-disk graph may have $\Theta(n^2)$ edges, it can be represented in $O(n)$ space: It suffices to store the points of S. Given any two points p and q of S, we can decide in $O(1)$ time if p and q are connected by an edge and, if so, compute its length $|pq|$. (If p and q are not connected by an edge, however, then a shortest-path computation is needed to compute $\delta(p, q)$.) Gao and Zhang [21] proved the following result:

Theorem 53.11

Let S be a set of n points in $\mathbb{R}^d$, and let $s > 1$ be a real number. Consider the unit-disk graph metric on S.

1. *If $d = 2$, then a WSPD for S with respect to s, consisting of $O(s^4 n \log n)$ pairs, can be computed in $O(s^4 n \log n)$ time.*
2. *If $d = 3$, then a WSPD for S, with respect to s, consisting of $O(n^{4/3})$ pairs, can be computed in $O(n^{4/3} \log^{O(1)} n)$ time.*
3. *If $d \geq 4$, then a WSPD for S, with respect to s, consisting of $O(n^{2-2/d})$ pairs, can be computed in $O(n^{2-2/d})$ time.*

Talwar [22] extended the WSPD to metric spaces whose *doubling dimension* is a constant. To define this notion, let (S, δ) be a metric space. A ball, with center $p \in S$ and radius R, is defined to be the set $\{q \in S : \delta(p, q) \leq R\}$. The *doubling parameter* of S is defined to be the smallest integer λ such that the following holds, for all real numbers $R > 0$: Every ball with radius R can be covered by λ balls of radius $R/2$. The doubling dimension of the metric space is defined to be $\log \lambda$. Observe that this generalizes Euclidean space $\mathbb{R}^d$, because the doubling dimension of $\mathbb{R}^d$ is proportional to d.

Many algorithms solving proximity problems in $\mathbb{R}^d$ are analyzed using a packing argument. If the doubling parameter λ is small, then, in many cases, a similar analysis can be used to efficiently solve these problems.

Talwar [22] showed how to compute a WSPD consisting of $O(s^{\log \lambda} n \log \Delta)$ pairs, where Δ is the aspect ratio, which is defined to be the ratio of the diameter and the closest pair distance. Har-Peled and Mendel [23] gave an improved construction and obtained the following result:

Theorem 53.12

Let (S, δ) be a metric space, let $n = |S|$, let λ be the doubling parameter of S, and let $s > 1$ be a real number. There exists a randomized algorithm that constructs, in $O(\lambda n \log n + s^{\log \lambda} n)$ expected time, a WSPD for S, with separation ratio s, consisting of $O(s^{\log \lambda} n)$ pairs.

Most results of the previous sections remain valid for metric spaces whose doubling parameter is bounded by a constant. Interestingly, Har-Peled and Mendel [23] show that this is not the case for the all-nearest neighbors problem: For every deterministic algorithm that solves this problem, there exists a metric space on n points and doubling parameter $\lambda \leq 3$, such that this algorithm computes all $\binom{n}{2}$ distances determined by these points.

References

[1] Preparata, F. P. and Shamos, M. I., *Computational Geometry: An Introduction*, Springer, Berlin, 1988.

[2] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O., *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, Berlin, 2000.

[3] Bern, M. and Eppstein, D., Approximation algorithms for geometric problems, in *Approximation Algorithms for NP-Hard Problems*, Hochbaum, D. S., Ed., PWS Publishing Company, Boston, MA, 1997, p. 296.

[4] Eppstein, D., Spanning trees and spanners, in *Handbook of Computational Geometry*, Sack, J.-R. and Urrutia, J., Eds., Elsevier Science, Amsterdam, 2000, p. 425.

[5] Smid, M., Closest-point problems in computational geometry, in *Handbook of Computational Geometry*, Sack, J.-R. and Urrutia, J., Eds., Elsevier Science, Amsterdam, 2000, p. 877.

[6] Callahan, P. B., Dealing with Higher Dimensions: The Well-Separated Pair Decomposition and its Applications, Ph.D. thesis, Department of Computer Science, Johns Hopkins University, Baltimore, MD, 1995.

[7] Callahan, P. B. and Kosaraju, S. R., A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields, *JACM*, 42, 67, 1995.

[8] Narasimhan, G. and Smid, M., *Geometric Spanner Networks*, Cambridge University Press, Cambridge, UK, 2007.

[9] Salowe, J. S., Constructing multidimensional spanner graphs, *Int. J. Comput. Geometry Appl.*, 1, 99, 1991.

[10] Salowe, J. S., Enumerating interdistances in space, *Int. J. Comput. Geometry Appl.*, 2, 49, 1992.

[11] Vaidya, P. M., Minimum spanning trees in k-dimensional space, *SIAM J. Comput.*, 17, 572, 1988.

[12] Vaidya, P. M., An $O(n \log n)$ algorithm for the all-nearest-neighbors problem, *Discrete Comput. Geometry*, 4, 101, 1989.

[13] Vaidya, P. M., A sparse graph almost as good as the complete graph on points in K dimensions, *Discrete Comput. Geometry*, 6, 369, 1991.

[14] Ramos, E. A., An optimal deterministic algorithm for computing the diameter of a three-dimensional point set, *Discrete Comput. Geometry*, 26, 233, 2001.

[15] Callahan, P. B. and Kosaraju, S. R., Faster algorithms for some geometric graph problems in higher dimensions, *Proc. of SODA*, 1993, p. 291.

[16] Agarwal, P. K., Edelsbrunner, H., Schwarzkopf, O., and Welzl, E., Euclidean minimum spanning trees and bichromatic closest pairs, *Discrete Comput. Geometry*, 6, 407, 1991.

[17] Erickson, J., On the relative complexities of some geometric problems, in *Proc. Canadian Conf. on Computational Geometry*, 1995, p. 85.

[18] Katz, M. J. and Sharir, M., An expander-based approach to geometric optimization, *SIAM J. Comput.*, 26, 1384, 1997.

[19] Bespamyatnikh, S. and Segal, M., Fast algorithms for approximating distances, *Algorithmica*, 33, 263, 2002.

[20] Bentley, J. L. and Saxe, J. B., Decomposable searching problems I: static-to-dynamic transformations, *J. Algorithms*, 1, 301, 1980.

[21] Gao, J. and Zhang, L., Well-separated pair decomposition for the unit-disk graph metric and its applications, *SIAM Journal on Computing*, 35, 151–169, 2005.

[22] Talwar, K., Bypassing the embedding: approximation schemes and compact representations of low dimensional metrics, *Proc. of STOC*, 2004, p. 281.

[23] Har-Peled, S. and Mendel, M., Fast construction of nets in low-dimensional metrics, and their applications, *SIAM Journal on Computing*, 35, 1148–1184, 2006.

54
Minimum-Edge Length Rectangular Partitions

Teofilo F. Gonzalez
University of California, Santa Barbara

Si Qing Zheng
University of Texas at Dallas

54.1 Introduction

In this chapter we discuss approximation algorithms for partitioning a rectangle with interior points. This problem is denoted by *RG-P*, where *RG* stands for *Rectangle* and *P* stands for *Points*. An instance of the *RG-P* problem is a set P of n points inside a rectangle R in the plane. A feasible solution is a *rectangular partition*, which consists of a set of (orthogonal) line segments that partition R into rectangles such that each of the n points in P is on a partitioning line segment. The objective of the *RG-P* problem is to find a rectangular partition whose line segments have least total length. The *RG-P* problem is an NP-hard problem [1].

A more general version of the problem is when R has interior rectilinear holes instead of points. This problem arises in VLSI design where it models the problem of partitioning a routing region into channels [2]. Approximation algorithms for this more general problem exist [3–6]. Levcopoulos' algorithms [4,5] are the ones with the smallest approximation ratio. His fastest algorithm [5] invokes as a subprocedure the divide-and-conquer algorithm for the *RG-P* problem developed by Gonzalez and Zheng [7], which is a preliminary version of the one discussed in this chapter.

The structure of an optimal rectangular partition can be very complex (Figure 54.1 [a]). A restricted version of the *RG-P* problem limits the set of feasible solutions to recursively defined partitions that at each level partition the instance into two subinstances of the *RG-P* problem by introducing a single line segment. A recursive partition with this property is given in Figure 54.1(b). For a given rectangle, such a line segment is called a *cut* of the rectangle in Ref. [7], and a *guillotine cut* in Ref. [8]. A rectangular partition such that at every level there is a guillotine cut is called a *guillotine partition.* By definition every guillotine partition is a rectangular partition, but the converse is not true. For example, the rectangular partition given in Figure 54.1(a) is not a guillotine partition. For the same instance, the structure of an optimal guillotine partition can be much simpler than the structure of an optimal rectangular partition. Furthermore, the concept of guillotine cut is useful for developing approximation algorithms for the *RG-P* problem using divide-and-conquer and dynamic programming techniques. The algorithm given in Ref. [7] finds an approximation to the *RG-P* problem by generating a suboptimal guillotine partition via a divide-and-conquer algorithm, whereas the algorithm given by Du et al. [8] finds an approximation to the *RG-P* problem by generating an optimal guillotine partition via dynamic programming. Gonzalez and Zheng's [7] algorithm takes $O(n \log n)$ time. Du et al.'s [8] algorithm takes $O(n^5)$ time to construct an optimal guillotine partition.

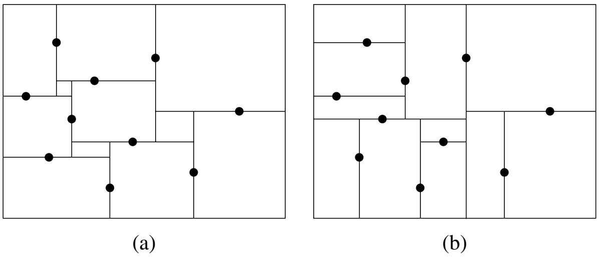


FIGURE 54.1 (a) Optimal rectangular partition. (b) Optimal guillotine partition.

On the other hand, Du et al. [8] showed that the length of an optimal guillotine partition is at most twice the length of an optimal rectangular partition for the *RG-P* problem, but the approximation ratio for the divide-and-conquer algorithm given in Ref. [7] is $3\sqrt{3}$. This ratio was reduced to 4 by using a slightly different divide-and-conquer procedure. A complex proof showing that the length of an optimal guillotine partition is within 1.75 times the length of an optimal rectangular partition is given by Gonzalez and Zheng [9]. Clearly, neither algorithm dominates the other when one takes into account both the time complexity and the approximation ratio.

It is important to note that optimal and near-optimal guillotine partitions are not the only way one can generate solutions with a constant approximation ratio for the *RG-P* problem. Gonzalez and Zheng [10] developed an algorithm with approximation ratio of 3 for the *RG-P* problem that generates a rectangular partition that is not necessarily a guillotine partition. Both the time and approximation ratio for this algorithm are between the ones of the two algorithms mentioned above.

The *RG-P* problem was generalized to d-dimensional Euclidean space. In this generalized problem, R is a d-box and P is a set of points in d-dimensional space, and the objective is to find a set of orthogonal hyperplane segments of least total $(d-1)$-volume that includes all points of P. This is the *d-dimensional RG-P problem.* Approximating an optimal d-box partition by suboptimal and optimal guillotine partitions based on divide-and-conquer and dynamic programming techniques has been established in Refs. [11,12], respectively.

In this chapter, we present two approximation algorithms for the *RG-P* problems. For an *RG-P* instance I, we use $E(I)$ to represent the set of line segments in the solution generated by our algorithm, and $E_{opt}(I)$ the set of line segments in an optimal rectangular partition. We use $L(S)$ to represent the length of the line segments in set S. We first present an $O(n \log n)$-time divide-and-conquer approximation algorithm that generates suboptimal guillotine partitions with total edge length within four times the one in an optimal rectangular partition, that is, for every I, $L(E(I)) \leq 4L(E_{opt}(I))$. We then present a simple proof that shows that an optimal guillotine partition has total edge length that is within two times the one in an optimal rectangular partition, that is, for every I, $L(E(I)) \leq 2L(E_{opt}(I))$. This proof is much simpler than the proof of Ref. [8] for the same bound. We also outline the main ideas of the complex proof in Ref. [9] that establishes that an optimal guillotine partition has edge length that is within 1.75 times the optimal rectangular partition value. One may improve Levcopoulos' [5] algorithm by replacing the algorithm in Ref. [7] by any of the above algorithms.

54.2 A Divide-and-Conquer Algorithm

An *RG-P* problem instance is given by $I = ((X, Y), P)$, where (X, Y) defines the rectangle R (with height Y and width X), and $P = \{p_1, p_2, \ldots, p_n\}$ is a nonempty set of points located inside R. Before we present our algorithm we define some terms and define a way to establish lower bounds for the length of rectangular partitions.

A *partial rectangular partition* is a partition of R into rectangles where not all the points in P are located along the partitioning line segments. Let Q be any partial rectangular partition for problem instance I. A subset of the points in P is said to be *assigned* to Q if the following three conditions are satisfied:

1. Every rectangle r in Q with at least one point inside it has one such point assigned to it.
2. A rectangle r in Q without points inside it may be assigned at most one point on one of its sides.
3. Two rectangles with a common boundary cannot have both their assigned points on their common boundary.

An assignment of the points in P to Q is denoted by $A(Q)$.

Every rectangle r with a point assigned to it is said to have *value*, $v(r)$, equal to the minimum of the height of r and the width of r. The assignment $A(Q)$ of the partial rectangular partition Q has value equal to the sum of the values of the rectangles that have an assigned point. This value is denoted by $v(A(Q))$.

We now establish that the edge length of a rectangular partition is at least equal to the value of any assignment for any partial rectangular partition of R.

Lemma 54.1

For every problem instance I, partial rectangular partition Q, and assignment $A(Q)$, a lower bound for $L(E(I))$ is given by $v(A(Q))$, that is, $v(A(Q)) \leq L(E(I))$. In particular, $v(A(Q)) \leq L(E_{OPT}(I))$.

Proof

Consider any rectangle r with one or more points in it. By definition one of these points is assigned to the rectangle. Clearly, any partition into rectangles must include line segments inside r with length greater or equal to the minimum of the height of r or the width of r. This is equal to the value $v(r)$.

Consider now any rectangle r without points inside, but with an assigned point. The point assigned to r must be located on one of its sides. Let us say it is on side s. By definition any rectangle with a common boundary to side s of rectangle r does not have its assigned point on this common boundary. Therefore, any rectangular partition that covers the assigned point of r must include line segments in r with length at least equal to the minimum of the height of r or the width of r. This is equal to the value $v(r)$.

Since $v(A(Q))$ is equal to the sum of the values of the rectangles that have been assigned a point, the above arguments establish that $v(A(Q)) \leq L(E(I))$, that is, $v(A(Q))$ is a lower bound for $L(E(I))$. □

We now define our divide-and-conquer procedure to generate a rectangular partition. From this rectangular partition we identify a partial rectangular partition and an assignment of a subset of the points P. Applying Lemma 54.1 we have a lower bound for an optimal rectangular partition. Then we show that the length of the edges in the solution generated is within four times the lower bound provided by the assignment.

Assume without loss of generality that we start with a nonempty problem instance such that $Y \leq X$. Procedure *DC* introduces a *mid-cut* or an *end-cut*, depending on whether or not the rectangle has points to the left and also to the right of the center of R. The cut is along a vertical line that partitions R into R_l and R_r. The set of points in P that are not part of the cut are partitioned into P_l and P_r depending on whether they are inside of R_l or R_r. A *mid-cut* is a vertical line segment that partitions R and includes the center of the rectangle. An *end-cut* is a vertical line segment that partitions R and includes either the “rightmost” or the “leftmost” points in P, depending whether all the points are located to the left or right of the center of R. Procedure *DC* is then applied recursively to the nonempty resulting subproblems, that is, $I_l = ((X_l, Y), R_l)$ and $I_r = ((X_r, Y), R_r)$, where X_l and X_r represent the length along the x-axis of the two resulting subinstances (I_l and I_r), respectively. Note that when a mid-cut is introduced it must be that both P_l and P_r end up being nonempty. For an end-cut at least one of these sets will be empty. When both sets of points are empty the cut is called *terminal end-cut.*

It is easy to verify that Figure 54.2 represents all the possible outcomes of one step in the recursive process of our procedure. A subinstance without interior points is represented by a rectangle filled with diagonal line segments.

Our lower bound function, $LB(I)$, is defined from a partial rectangular partition of the rectangular partition generated by Procedure *DC*. The partial rectangular partition is the rectangular partition generated

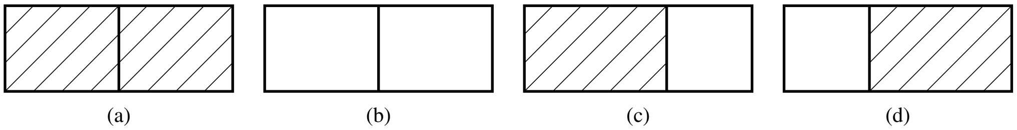


FIGURE 54.2 Subinstances generated by Procedure *DC*.

by Procedure *DC*, except that all the terminal end-cuts are not included. The association for the partial rectangular partition is defined as follows. Every rectangle with points inside it is assigned one of the points inside it. Rectangles without points inside them resulting from a nonterminal end-cut have one of the points along the end-cut assigned to them. By Lemma 54.1, the value of this assignment is a lower bound for the length of an optimal rectangular partition. A recursive definition for the value of the above assignment is given by the following recurrence relation:

$$LB(I) = \begin{cases} Y & \text{(a) A terminal end-cut is introduced, } P_l = \emptyset \text{ and } P_r = \emptyset. \\ LB(I_l) + LB(I_r) & \text{(b) A mid-cut is introduced, } P_l \neq \emptyset \text{ and } P_r \neq \emptyset. \\ \min\{Y, X_l\} + LB(I_r) & \text{(c) A nonterminal end-cut is introduced, } P_l = \emptyset \text{ and } P_r \neq \emptyset \\ LB(I_l) + \min\{Y, X_r\} & \text{(d) A nonterminal end-cut is introduced, } P_l \neq \emptyset \text{ and } P_r = \emptyset. \end{cases}$$

Let $L(E_{DC}(I))$ denote the total length of the set $E_{DC}(I)$ of line segments introduced by Procedure *DC*. We define $USE(I)$ to be the length of the line segments introduced directly by Procedure $DC(I)$, but not by the recursive invocations, that is, when $P = \emptyset$ then $USE(I) = 0$; otherwise, $USE(I) = L(E_{DC}(I)) - L(E_{DC}(I_l)) - L(E_{DC}(I_r))$.

Assume $X \geq Y$. A problem instance $I = ((X, Y), P)$ is said to be *regular* if $X \leq 2Y$, and *irregular* otherwise (i.e., $X > 2Y$). We define the *carry function* C for a problem instance I as

$$C(\ I\ ) \ = \ \begin{cases} 3Y & \text{if } I \text{ is irregular} \\ X + Y & \text{if } I \text{ is regular} \end{cases}$$

One may visualize the analysis of our approximation algorithm as follows. Whenever a line segment (*mid-cut* or *end-cut*) is introduced by Procedure *DC* it is colored red, and when a lower bound from $LB(I)$ is "identified" we draw an "invisible" blue line segment with such length. Our budget is four times the length of the blue line segments, which we must use to pay for all the red line segments. In other words, the idea is to bound the sum of the length of all the red segments by four times the one of the blue segments. The length of the red line segments introduced at previous recursive invocations of Procedure *DC* which have not yet been accounted by previously identified blue segments is bounded above by the carry function C, defined above. In other words, the carry value is the maximum edge length (length of red segments) we could possibly owe at this point. Before proving our result, we establish some preliminary bounds.

Lemma 54.2

For any problem instance I such that $X \geq Y, 2Y \leq C(I) \leq 3Y$.

Proof

The proof follow from the definition of the carry function. □

Lemma 54.3

For every problem instance I, $L(E_{DC}(I)) + C(I) \leq 4LB(I)$.

Proof

The proof is by contradiction. Let I be a problem instance with the least number of points P that does not satisfy the lemma, that is,

$$L(E_{DC}(I)) + C(I) > 4LB(I) \tag{54.1}$$

Assume without loss of generality that $Y \leq X$. There are three cases depending on the cut introduced by Procedure *DC* when it is initially invoked with I.

Case 1. The procedure introduces a terminal end-cut.
Since $Y \leq X$, and $P_l = P_r = \emptyset$, we know that $LB(I) = Y$. From the way the procedure operates we know $L(E_{DC}(I)) = USE(I) = Y$. Substituting into Eq. (54.1), we know that $C(I) > 3Y$. But this contradicts Lemma 54.2. So it cannot be that the algorithm introduces a terminal end-cut when presented with instance I.

Case 2. The procedure introduces a mid-cut.
Since a *mid-cut* is introduced, both P_l and P_r must be nonempty and since both I_l and I_r have fewer points than P, we know they satisfy the conditions of the lemma. Combining the conditions of the lemma for I_l and I_r we know that

$$L(E_{DC}(I_l)) + L(E_{DC}(I_r)) + C(I_l) + C(I_r) \leq 4LB(I_l) + 4LB(I_r)$$

By definition, $L(E_{DC}(I)) = L(E_{DC}(I_l)) + L(E_{DC}(I_r)) + USE(I)$ and $LB(I) = LB(I_l) + LB(I_r)$. Since $Y \leq X$ we know $USE(I) = Y$. Substituting these equations into Eq. (54.1) we have

$$Y + C(I) > C(I_l) + C(I_r) \tag{54.2}$$

There are two cases depending on whether I is regular or irregular.

Subcase 2.1. I is regular.
By definition $C(I) = X + Y$. Substituting into Eq. (54.2),

$$Y + C(I) = X + 2Y > C(I_l) + C(I_r)$$

Since I is regular and the procedure introduces a mid-cut, both I_1 and I_2 must also be regular. Therefore, $X + 2Y = C(I_l) + C(I_r)$. A contradiction. So it cannot be that I is regular.

Subcase 1.2. I is irregular.
Since $Y \leq X$ and I is irregular, we know that $Y + C(I) = 4Y$. Substituting into Eq. (54.2) we have $4Y > C(I_l) + C(I_r)$. Since I is irregular and the algorithm introduces a mid-cut, it must be that $X_l = X_r \geq Y$. But by Lemma 54.2, $C(I_l) \geq 2Y$ or $C(I_r) \geq 2Y$. A contradiction. So it cannot be that the procedure introduces a min-cut.

Case 3. The procedure introduces a nonterminal end-cut.
When a nonterminal end-cut is introduced, exactly one of the two resulting subproblems has no interior points (Figure 54.2 [c] and [d]). Assume without loss of generality that I_r has no interior points. From the lower bound function and the procedure we know that

$$LB(I) = LB(I_l) + \min\{Y, X_r\} \text{ and } L(E_{DC}(I)) = L(E_{DC}(I_l)) + USE(I)$$

Since instance I_l has fewer points than I, it then follows that $L(E_{DC}(I_l)) + C(I_l) \leq 4LB(I_l)$. Clearly, $USE(I) = Y$. Substituting these inequalities into Eq. (54.1) we know that

$$Y + C(I) > C(I_l) + 4\min\{Y, X_r\} \tag{54.3}$$

By Lemma 54.2 we know $C(I) \leq 3Y$ and $C(I_l) > 0$. So Eq. (54.3) is $4Y > 4\min\{Y, X_r\}$. Therefore, it cannot be that $Y \leq X_r$ as otherwise there is a contradiction. It must then be that $X_r < Y$. Substituting into Eq. (54.3)

$$Y + C(I) > C(I_l) + 4X_r \tag{54.4}$$

Instance I is regular because $X_r < Y$ and $X_r \geq \frac{X}{2}$ implies $X < 2Y$. Substituting $C(I) = X + Y$ into Eq. (54.4) we know

$$X + 2Y > C(I_l) + 4X_r \tag{54.5}$$

If I_l is regular, then substituting $C(I_l)$ into Eq. (54.5) we know that $X + 2Y > X_l + Y + 4X_r$. Since $X_l + X_r = X$, Eq. (54.5) becomes $Y > 3X_r$. But we know that $X_r \geq X/2$ and $X \geq Y$. So $X_r \geq Y/2$. A contradiction. So it must be that I_l is irregular.

In contrast, if I_l is irregular, then since $X_l \leq Y$ substituting $C(I_l)$ into Eq. (54.5) we know that $X + 2Y > 3X_l + 4X_r$. Since $X_l + X_r = X$, we know that $2Y > 2X + X_r$. But we know that $X \geq Y$ and $X_r > 0$. A contradiction.

This completes the proof of this case and the lemma. □

We establish the main result (Theorem 54.1), which shows that the approximation bound is tight (Theorem 54.2), and explain implementation details needed to establish the time complexity bound for procedure *DC* (Theorem 54.3).

Theorem 54.1

For any instance of the RG-P problem, algorithm DC generates a solution such that $L(E_{DC}(I)) \leq 4\,L(E_{opt}(I))$.

Proof

The proof follows from Lemmas 54.2 and 54.3. □

We now show that the approximation bound is asymptotically tight, that is, $L(E_{DC}(I))$ is about $4L(E_{opt}(I))$. The problem instance we use to establish this result has the property that $LB(I) = L(E_{opt}(I))$. Our approach is a standard one that begins with small problem instances and then combines them to build larger ones. As the problem instances become larger, the approximation ratio for the solution generated by Procedure *DC* increases. The analysis just needs to take into consideration a few steps performed by the procedure and the previous analysis for the smaller problem instances.

We define problem instances P_i, for $i \geq 0$ as follows: Problem instance P_i consists of a rectangle of size 2^i by 2^i. The instance P_1 contains two points. Figure 54.3(a) and Figure 54.3(b) depicts $E_{DC}(P_1)$ and $E_{opt}(P_1)$, respectively. Clearly, $L(E_{DC}(P_1)) = 4$ and $L(E_{opt}(P_1)) = 2$. In this case the approximation ratio is 2. Instance P_2 is a combination of four instances of P_1. Figure 54.3(c) and Figure 54.3(d) depicts $E_{DC}(P_2)$ and $E_{opt}(P_2)$, respectively. Clearly, $L(E_{DC}(P_2)) = 2^3 + 4L(E_{DC}(P_1)) = 24$ and $L(E_{opt}(P_2)) = 4L(E_{opt}(P_1)) = 8$. The ratio is 3. Problem instance P_3 combines four instances of P_2 as shown in Figure 54.3(c). Figure 54.3(e) and Figure 54.3(f) depicts $E_{DC}(P_3)$ and $E_{opt}(P_3)$, respectively. Clearly, $L(E_{DC}(P_3)) = 2^4 + 4L(E_{DC}(P_2)) = 112$ and $L(E_{opt}(P_3)) = 4L(E_{opt}(P_2)) = 32$. The ratio is $112/32 = 3.5$. To construct P_i we apply the same combination using P_{i-1}. Note that when applying our procedure to P_i it always introduces mid-cuts, except when presented P_0 in which case it introduced a terminal end-cut. It is simple to see that our approximation algorithm introduces

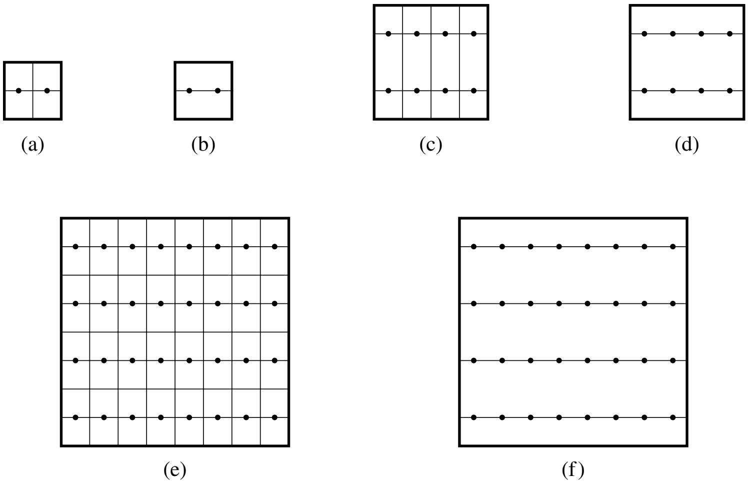


FIGURE 54.3 (a) $E_{DC}(P_1)$, (b) $E_{OPT}(P_1)$, (c) $E_{DC}(P_2)$, (d) $E_{OPT}(P_2)$, (e) $E_{DC}(P_3)$, and (f) $E_{OPT}(P_3)$.

cuts with length $L(E_{DC}(P_i)) = 2^{i+1} + 4L(E_{DC}(P_{i-1}))$, for $i > 1$ and $L(E_{DC}(P_1)) = 4$. An optimal solution, in this case, is identical to the lower bound function which is $L(E_{opt}(P_i)) = 4L(E_{opt}(P_{i-1}))$, for $i > 1$ and $L(E_{opt}(P_1)) = 2$.

Solving the above recurrence relations we know that

$$L(E_{DC}(P_i)) = \sum_{j=0}^{i-2} 2^{2i-j-1} + 2^{2i-2} L(E_{DC}(P_1)) = 2^{2i+1} - 2^{i+1}$$

and

$$L(E_{OPT}(P_i)) = 2^{2(i-1)} L(E_{OPT}(P_1)) = 2^{2i-1}$$

Therefore the approximation ratio is $L(E_{DC}(P_i))/L(E_{opt}(Pi)) = 4 - 1/2^{i-2}$.

Theorem 54.2

There are problem instances for which procedure DC generates a solution such that $L(E_{DC}(I))$ tends to $4L(E_{opt}(I))$.

Proof

By the above discussion. □

Procedure *DC* can be easily modified so that for problem instances with "many" points along the same line it will introduce a line to cover all the points provided that the line segment has length close to Y. For the problem instance given above the modified algorithm will generate a better solution decreasing substantially the approximation ratio. However, as pointed out in Ref. [11], there are problem instances for which the modified procedure will generate solutions with edge length of about $4LB(I)$. The idea is to perturb the points slightly. This way no two points will belong to the same vertical or horizontal line.

The time complexity $T(n)$ for Procedure *DC* when operating on an instance with $n = |P|$ points is given by the recurrence relation $T(n) \leq T(n-i-1) + T(i) + cn$, for $1 \leq i < n$ and some constant c. However, it is possible to implement the procedure so that it takes $O(n \log\, n)$ time [5]. In what follows, we briefly describe one of the two implementations given in Ref. [5] with the $O(n \log\, n)$ bound. To simplify the presentation assume that no two points can be covered by the same vertical or horizontal line. The idea is to change the procedure so that the time complexity term cn becomes $cf(\min\{i, n-i-1\})$, for some function $f()$ that we specify below. Note that this requires a preprocessing step that takes $O(n \log n)$ time. For certain functions $f()$ this reduces the overall time complexity bound to $O(n \log\, n)$.

In the preprocessing step we create a multilinked structure in which there is a record (data node) for each point. The record contains the x- and y-coordinate values of the point. We also include the rank of each point with respect to their x and y values. For example, if the rank of a point is (i, j) then it is the ith smallest point with respect to its x value, and the jth smallest point with respect to its y value. Note that this ranking is with respect to the initial set of points. We also have all of these records in two circular lists, one sorted with respect to the x values and the other sorted with respect to the y values. It is simple to see that this multilinked structure can be constructed in $O(n \log\, n)$ time.

In each recursive invocation of procedure *DC* we need to construct P_l and P_r from P, and assume that $X \geq Y$. This construction can be implemented to take $O(\min\{|P_l|, |P_r|\})$ by scanning the doubly linked lists for the x values from both ends (alternating one step from each end) until all the points at one end are P_l and the other ones are on P_r. Assume that $n \geq |P| = m > |P_l| = m - q \geq |P_r| = q$, and $1 < q \leq m/2$. Clearly, the above procedure can be used to identify the points in P_r and then remove the points in P_r from P in $O(q)$ time. The remaining points are P_l and are represented according to our structure. But now the problem is that we need to construct a multilinked data structure for the points in P_l. These points are already sorted by their x values, but not by their y values. The algorithm given in Ref. [13] can sort any q integers in the range $[1, n]$ in $O(q \log \log_q\, n)$ time. Once we have sorted them we can construct the multilinked data structure for P_l. This takes $O(q \log \log_q n)$ time. Let $T_n(m)$ be the total time required by procedure *DC* when the initial invocation involved a problem with n points and we now have a problem with m points. Clearly, $T_n(1) = O(1)$ and $T_n(m) = \max\{O(q \log \log_q n) + T_n(q) + T_n(m - q) | 1 \leq q \leq m/2\}$ for $m > 1$. By the analysis of Ref. [5], $T_n(n) = O(n \log n)$. This result is summarized in the following theorem:

Theorem 54.3

Procedure DC and its preprocessing step can be implemented to take $O(n \log n)$ time.

Proof

By the above discussion. □

54.3 Dynamic Programming Approach

The divide-and-conquer algorithm *DC* presented in the previous section introduces guillotine cuts by following a set of simple rules, which makes the algorithm run very efficiently. But such guillotine cuts do not form an optimal guillotine partition. It seems natural that using an optimal guillotine partition would generate a better solution. But how fast can one generate an optimal guillotine partition? By the recursive nature of guillotine partitions, it is possible to construct an optimal guillotine partition in polynomial time. In this section we analyze an approximation algorithm based on this approach. As we shall see the algorithm has a smaller approximation ratio, but it takes longer to generate its solution.

54.3.1 Algorithm

Let $I = (R = (X, Y), P)$ be any problem instance and let the x-interval and y-interval define the rectangle $R_{x,y}$, which is part of rectangle R. We use $g(R_{x,y})$ to represent the length of an optimal guillotine partition for $R_{x,y}$.

First it is important to establish that there is always an optimal guillotine partition such that all its line segments include at least one point from P inside them (excluding those at its endpoints). This is based on the observation that given any optimal partition that does not satisfy this property it can either be transformed to one that does satisfy the property or one can establish that it is not an optimal guillotine partition. The idea is to move each horizontal (vertical) guillotine cut without points from P inside them either to the left or right (up or down) without increasing the total edge length. Note that when the above operation is perfomed some vertical (horizontal) segments need to be extended and some need to be retracted to preserve a guillotine partition. If the total edge length decreases then we know that it is not an optimal guillotine partition and when it remains unchanged after making all the transformations it becomes an optimal guillotine partition in which all its guillotine cuts include at least one point from P.

By applying the above argument we know that for any rectangle $R_{x,y}$ one can easily compute $g(R_{x,y})$ recursively by selecting the best solution obtained by trying all $2n$ guillotine cuts (that include a point from P) and then solving recursively the two resulting problem instances. It is simple to show that there are $O(n^4)$ different g values that need to be computed. By using dynamic programming and the above recurrence relation the length of an optimal guillotine partition can be constructed in $O(n^5)$ time. By recording at each step a guillotine cut forming an optimal solution and using the g values, an optimal guillotine partition can be easily constructed within the same time complexity bound. The following theorem follows from the above discussion:

Theorem 54.4

An optimal guillotine partition for the RG-P problem can be constructed in $O(n^5)$ time.

54.3.2 Approximation Bound

The set of line segments in a feasible rectangular partition of I is denoted by $E(I)$, and a set of line segments in a minimum-length guillotine partition is denoted by $E_G(I)$). We use $L(E)$ to represent the length of the edges in a rectangular partition E. In what follows we show that $L(E_G(I)) \leq 2L(E(I))$ by introducing a set of vertical and horizontal line segments $A(I)$ such that $E(I) \cup A(I)$ is a guillotine partition and $L(E(I) \cup A(I)) \leq 2L(E(I))$. The bound follows from the fact that $L(E_G(I)) \leq L(E(I) \cup A(I))$. Our main result follows from the application of this result to $E(I) = E_{opt}(I)$. The set $A(I)$ of additional segments are introduced by Procedure *TR1*. Before we present this procedure and in its analysis we define some terms.

A horizontal (vertical) line segment that partitions rectangle R into two rectangles is called a *horizontal (vertical) cut.* We say that $E(I)$ has a *horizontal guillotine cut,* if there is a horizontal cut, l, such that $L(E(I) \cap l) = X$. A rectangular partition $E(I)$ has a *half horizontal overlapping cut* if there is a horizontal cut l such that $L(E(I) \cap l) \geq 0.5X$. *Vertical guillotine cuts* are defined similarly.

Suppose R is partitioned by a vertical or horizontal cut into two rectangles, R_l and R_r. With respect to this partition we define $E(I_l)$ and $E(I_r)$ as the set of line segments of $E(I)$ inside R_l and R_r, respectively. We use $E_h(I)$ ($E_v(I)$) to denote all the horizontal (vertical) line segments in $E(I)$. With respect to $A(I)$ we define similarly $A_h(I)$ and $A_v(I)$.

Assume that $E(I)$ is nonempty. The idea behind Procedure *TR1* is to either introduce horizontal or vertical line segments at each step and then apply recursively the procedure to the nonempty problem instances. When a horizontal line segment is introduced, it is added along a half horizontal overlapping cut. The two resulting problems will not have any of these segments inside them. So at this step the added horizontal segments have length at most equal to the ones of the half horizontal overlapping cut.

Since there are problem instances without half horizontal overlapping cuts, Procedure *TR1* checks to see if there is a vertical guillotine cut. In this case we just partition the rectangle along this cut. Clearly, there are no additional line segments introduced in this case.

There are rectangular partitions without a half horizontal overlapping cut or a vertical guillotine cut. In this case, Procedure *TR1* introduces a *mid-cut*, which is just a vertical line that partitions the rectangle along its center. The problem now is that this mid-cut does not necessarily overlap with any of the segments in $E_v(I)$. Our approach is to remember this fact and later on identify a set of line segments in $E_v(I)$ that will account for the length of this mid-cut. To remember this fact we will color the right side of rectangle R_l. As we proceed in the recursive process different parts of this colored side will be inherited by smaller rectangles that will get their right side colored. At some point later on when we pay for the segment represented by a colored right side of a rectangle, it will no longer appear in recursive calls resulting from the one for this rectangle. The budget in this case is two times the total length of the vertical line segments in $E_v(I)$. This budget should be enough to pay for the total length of the vertical line segments introduced during the transformation.

To be able to show that $A_v(I) \leq E_v(I)$ it must be that the rectangles in the terminal recursive calls will not have their right side colored. Before we establish this result, we formally present Procedure *TR1* that defines the way colors are inherited in the recursive calls.

```
Procedure TR1($I = (R = (X, Y), E(I))$);
    case
      :$E(I)$ is empty: return;
      :There is a half horizontal overlapping cut in $E(I)$:
         Partition the rectangle $R$ along one such cut;
         If the right side of $R$ is colored, the right sides of rectangles $R_l$ and $R_r$ remain colored;
      :There is a vertical guillotine cut in $E(I)$:
         Partition $R$ along one such cut;
         Remove the color, if any, of the right side of rectangle $R_r$;
      :else:
         Partition $R$ by a vertical cut intersecting the center of $R$;// introduce a mid-cut
         If the right side of $R$ is colored, the right side of $R_r$ remains colored;
         Color the right side of $R_l$;
    endcase
  Apply TR1 recursively to ($I_l = (R_l, E(I_l))$);
  Apply TR1 recursively to ($I_r = (R_r, E(I_r))$);
end of Procedure TR1
```

It is important to remember that Procedure *TR1* is only used to establish our approximation bound. The right side of rectangle R is not colored in the initial invocation to Procedure *TR1*. Before establishing the approximation ratio of 2 in Theorem 54.5, we prove the following lemmas:

Lemma 54.4

Every invocation of Procedure TR1 $(I, E(I))$ satisfies the following conditions:

(a) if $E(I)$ is empty, then the right side of R is not colored, and

(b) if the right side of R is colored, then $E(I)$ does not have a horizontal guillotine cut.

Proof

Initially the right side of R is not colored so the the first invocation $(I, E(I))$ satisfies conditions (a) and (b). We now show that if upon entrance to the procedure the conditions (a) and (b) are satisfied, then the invocations made directly from it will also satisfy (a) and (b). There are three cases depending on the type of cut introduced by procedure *TR1*.

Case 1. Procedure *TR1* partitions R along a half horizontal overlapping cut.
First consider the subcase when $E(I)$ has a horizontal guillotine cut. From (b) we know that the right side of R is not colored, and the algorithm does not color the right side of R_l or R_r. Therefore the two invocations made directly from this call satisfy properties (a) and (b). In contrast, when $E(I)$ does not have a horizontal guillotine cut, then the right side of R may be colored. Since $E(I)$ does not have a horizontal guillotine cut, we know that there is at least one vertical line segment on each side of the half horizontal overlapping cut, so $E(I_l)$ and $E(I_r)$ must be nonempty. Since neither of these two partitions has a horizontal guillotine cut, each invocation made directly by our procedure satisfies properties (a) and (b). This completes the proof for this case.

Case 2. Procedure *TR1* partitions R along a vertical guillotine cut.
Since the right side of R_l and R_r end up uncolored; then the invocations made directly by the procedure satisfy (a) and (b). This completes the proof of this case.

Case 3. Procedure *TR1* introduces a mid-cut.
Since $E(I)$ does not have a half horizontal overlapping cut and there is no vertical guillotine cut, then each of the resulting rectangular partitions has at least one vertical line segment and there are no horizontal guillotine cuts in the two resulting problems. Therefore, both of the resulting problem instances are not empty and do not have a horizontal guillotine cut. This implies that both problem instances satisfy (a) and (b). This completes the proof for this case and the lemma. □

Lemma 54.5

For any nonempty rectangular partition $E(I)$ of any instance I of the RG-P problem, procedure TR1 generates a set $A(I)$ of line segments such that $L(A_h(I)) \leq L(E_h(I))$, and $L(A_v(I)) \leq L(E_v(I))$.

Proof

First we show that $L(A_h(I)) \leq L(E_h(I))$. This is simple to prove because horizontal cuts are only introduced over half horizontal overlapping cuts. Each time a horizontal cut is introduced the segments added to $A_h(I)$ have length that is at most equal to the length of the segments in the half horizontal cut in $E_h(I)$. Since these line segments are located on the boundary of the two resulting instances, these line segments will not account for other segments in $A_h(I)$ and thus $L(A_h(I)) \leq L(E_h(I))$.

Let us now establish that $L(A_v(I)) \leq L(E_v(I))$. It is simple to verify that Procedure *TR1* does not color a side more than once, and all empty rectangular partitions do not have their sides colored (Lemma 54.4). Every invocation of Procedure *TR1* with a nonempty rectangular partition R generates two problem instances whose total length of their colored right sides is at most the length of the right side of R, if it is colored, plus the length of the vertical line segment introduced. The only exception is when the procedure introduces a cut along an existing vertical guillotine cut. In this case if the right side of R is colored, the right sides of two resulting partitions will not be colored. So the length of a line segment previously introduced

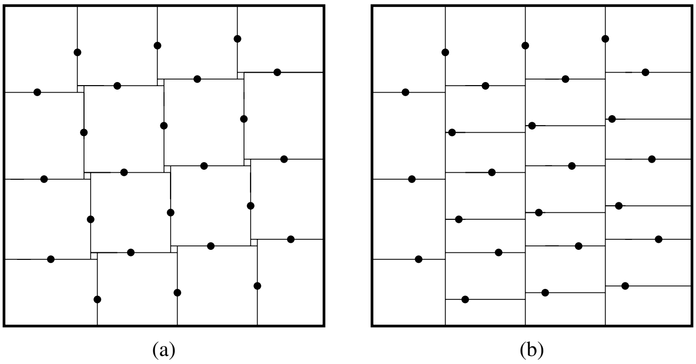


FIGURE 54.4 (a) Optimal rectangular partition. (b) Optimal guillotine partition.

in $A(I)$, which is recorded by the fact that the right side of R is colored, is charged to the existing guillotine cut and such cut will not be charged another segment again. Therefore, $L(A_v(I)) \leq L(E_v(I))$. This concludes the proof of the lemma. □

Theorem 54.5

The length of an optimal guillotine partition is at most twice the length of an optimal rectangular partition, that is, $L(E_G(I)) \leq 2L(E_{opt}(I))$.

Proof

Apply procedure *TR1* to any optimal rectangular partition $E(I) = E_{opt}$. By Lemma 54.5 we know that $L(E(I) \cup A(I)) \leq 2L(E_h(I)) + 2L(E_v(I)) = 2L(E(I))$. Since $E(I) \cup A(I)$ is a guillotine partition, $L(E_G(I)) \leq L(E(I) \cup A(I))$. Hence, $L(E_G(I)) \leq 2L(E(I)) = 2L(E_{opt}(I))$. □

It is simple to find a problem instance I such that $L(E_G(I))$ is about $1.5L(E_{opt}(I))$ [8]. One of such problem instances has the distribution of points shown in Figure 54.4. As the number of points increases, the ratio $\frac{L(E_G(I))}{L(E_{opt}(I))}$ approaches 1.5.

54.3.3 Improved Approximation Bound

In this section, we describe the idea behind the complex proof given in Ref. [9] that establishes the fact that $L(E_G(I)) \leq 1.75L(E_{opt}(I))$. The proof is based on a recursive transformation procedure *TR2* that, when performed on any rectangular partition $E(I)$, generates a set $A(I)$ of line segments such that $E(I) \cup A(I)$ forms a guillotine partition (of course $A(I) \cap E(I) = \emptyset$). Without loss of generality, assume that $L(E_v(I)) \leq L(E_h(I))$. The transformation is performed in such a way that

$$L(A_v(I)) \leq L(E_v(I)) \tag{54.6}$$

and

$$L(A_h(I)) \leq 0.5L(E_h(I)) \tag{54.7}$$

Then, we have

$$\begin{aligned} L(E_G(I)) &\leq L(A(I) \cup E(I)) \\ &= L(A_h(I)) + L(A_v(I)) + L(E(I)) \\ &\leq 0.5L(E_h(I)) + L(E_v(I)) + L(E(I)) \\ &= 0.5L(E_v(I)) + 1.5L(E(I)) \\ &\leq 1.75L(E(I)) \end{aligned}$$

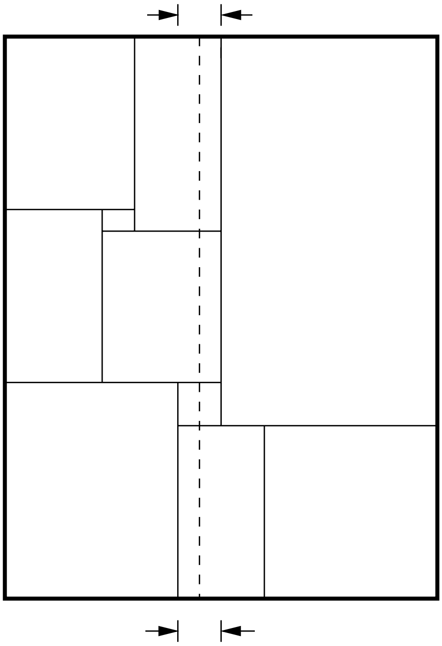


FIGURE 54.5 A vertically separable $E(I)$. $E(I)$ has no guillotine cut. The broken line segment is a vertical through cut of $E(I)$. In fact, any vertical cut in the region marked by the vertical segments outside of rectangular boundary is a vertical through cut for $E(I)$.

To satisfy Eq. (54.6), the notion of *vertical separability* is introduced. Denote the x-coordinate of a vertical segment l by $x(l)$. A vertical cut l is *left* (*right*) *covered by* $E_v(I)$ if for every point p on l there exists a line segment l' in $E_v(I)$ such that $x(l') \leq x(l)$ $(x(l') \geq x(l))$, and there is a point p' on l' with $x(p') = x(p)$. A vertical cut is called a *vertical through cut* if it is both left and right covered by $E_v(I)$. The set of segments $E(I)$ is said *vertically separable* if there exists at least one vertical through cut. Note that a vertical guillotine cut is also a vertical through cut, but the converse is not necessarily true. Figure 54.5 shows a vertically separable $E(I)$ and a vertical through cut. When a new vertical line segment is introduced by *TR2*, it is ensured to be a portion of a vertical through cut.

To satisfy Eq. (54.7), horizontal segments are carefully introduced by *TR2* according to the structure of $E(I)$. When there is neither guillotine cut and nor vertical through cut in $E(I)$, a three-step subprocedure *HVH_CUT* is invoked to introduce new segments. Procedure *TR2* is given below

```
Procedure TR2($E(I)$);
    case
      :$E(I)$ is empty:
         return;
      :$E(I)$ has a guillotine cut $l$:
         Partition $E(I)$ along $l$ into $E(I_1)$ and $E(I_2)$;
         Recursively apply TR2 to $E(I_1)$ and $E(I_2)$;
      :$E(I)$ is vertically separable:
         Let $l$ be any vertical through cut;
         Partition $E(I)$ along $l$ into $E(I_1)$ and $E(I_2)$;
         Recursively apply TR2 to $E(I_1)$ and $E(I_2)$;
      :else:
         Use Procedure HVH_CUT to partition $E(I)$ into $E(I_1)$, $E(I_2)$, $\cdots$, $E(I_{q+1})$;
         Recursively apply TR2 to each $E(I_i)$;
    endcase
end of Procedure TR2
```

Procedure *HVH_CUT* is outlined below:

```
Procedure HVH_CUT(E(I));
  1. Introduce a carefully selected set of $q$ horizontal cuts that partition $E(I)$ into $q + 1$ vertically
     separable rectangular subpartitions;
  2. In each resulting partition of Step 1, introduce either the leftmost or the rightmost vertical through
     cut;
  3. Divide each resulting partition of Step 2 into two rectangular subpartitions by a horizontal guillotine
     cut if such a cut exists;
end of Procedure HVH_CUT
```

Let $H(I)$ be the set of horizontal cuts introduced in Step 1, $V(I)$ the set of vertical through cuts chosen in Step 2, and $H_3(I)$ the horizontal guillotine cuts found in Step 3. Define $H_1(I) = H(I) \cap E_h(I)$ and $H_1'(I) = H(I) - E_h(I)$. Note that $H_3(I) \subset E_h(I)$. The sets $H(I)$ and $V(I)$ are selected in such a way that

$$L(H_1'(I)) \leq 0.5qX \tag{54.8}$$

and

$$L(H_1(I) \cup H_3(I)) \geq qX \tag{54.9}$$

It is quite complex to establish that such $H(I)$ and $V(I)$ always exist [9]. We omit additional details of the procedure *HVH_CUT* and the related proofs. Vertical though cuts introduced by *TR2* are carefully selected to satisfy Eq. (54.6) and ensure Eq. (54.9). Since all horizontal cuts introduced by *HVH_CUT* satisfy Eq. (54.8) and Eq. (54.9), and all horizontal cuts that are not introduced by invocations to *HVH_CUT* are horizontal guillotine cuts in their respective subrectangular boundaries, Eq. (54.7) is satisfied. Consequently, $L(E_G(I)) \leq 1.75L(E(I))$. Since $E(I)$ is any arbitrary rectangular partition, we have $L(E_G(I)) \leq 1.75L(E_{opt}(I))$. The next theorem sums up the discussion.

Theorem 54.6

The length of an optimal guillotine partition is at most 1.75 times the length of an optimal rectangular partition, that is, $L(E_G(I)) \leq 1.75L(E_{opt}(I))$.

Proof

The full details of the proof, whose outline is given above, appears in Ref. [9]. □

54.4 Concluding Remarks

In this chapter, we considered the *RG-P* problem. We presented a fast divide-and-conquer approximation algorithm with approximation ratio 4. This ratio is smaller than the $3 + \sqrt{3}$ ratio of a similar divide-and-conquer approximation algorithm given [7]. We also examined in detail the approach of approximating optimal rectangular partitions via optimal guillotine partitions. Optimal guillotine partitions can be constructed in polynomial time using dynamic programming [3]. For this approach we presented a proof that the approximation ratio is at most 2. Our proof is simpler than the proof for the same bound given in Ref. [3]. We presented the idea behind a complex proof given in Ref. [9], which establishes that the approximation ratio is at most 1.75 when approximating optimal rectangular partitions via optimal guillotine partitions. Both proofs are based on the technique of recursively transforming a rectangular partition into a guillotine partition by introducing additional line segments. The difference is that the additional segments in the transformation for the approximation ratio of 1.75 are selected more carefully than those introduced by the transformation for the bound of 2. Whether or not one can further reduce this bound remains a challenging open problem.

For the *RG-P* problem, the partitions obtained by the divide-and-conquer algorithms (the one of Ref. [7] and the one presented in this chapter) and the dynamic programming algorithm are guillotine partitions formed by recursive guillotine cuts. The approach examined in this chapter can be referred to as approximating optimal rectangular partitions via optimal and suboptimal guillotine partitions. The first approximation algorithm for this problem based on suboptimal guillotine partitions appeared in Ref. [7]. Subsequently, optimal guillotine partitions were used in Ref. [8]. Both of these approaches were generalized to multidimensional space in Refs. [11,12].

The term "guillotine cut" was introduced in the 1960s in the context of cutting stock problems. In the context of rectangular partitions it was first used in Ref. [8]. As pointed out in Ref. [14], the concept of guillotine partition has been generalized into a powerful general approximation paradigm for solving optimization problems in different settings. The guillotine partition algorithms for the *RG-P* problem were among the first that manifested the power of this paradigm.

References

[1] Lingas, A., Pinter, R. Y., Rivest, R. L., and Shamir, A., Minimum edge length partitioning of rectilinear polygons, in *Proc. 20th Allerton Conf. on Comm., Cont., and Comput.*, Monticello, Illinois, 1982.

[2] Rivest, R. L., The "PI" (placement and interconnect) system, in *Proc. Design Automation Conf.*, 1982.

[3] Du, D. Z. and Chen Y. M., On Fast Heuristics for Minimum Edge Length Rectangular Partition, Technical report, MSRI 03618–86, 1986.

[4] Levcopoulos, C., Minimum length and thickest–first rectangular partitions of polygons, *Proc. 23rd Allerton Conf. on Communication, Control and Computing,* Monticello, Illinois, 1985.

[5] Levcopoulos, C., Fast heuristics for minimum length rectangular partitions of polygons, *Proc. 2nd ACM Symp. on Computational Geometry,* 1986.

[6] Lingas, A., Heuristics for minimum edge length rectangular partitions of rectilinear figures, *Proc. 6th GI–Conf.,* Lecture Notes in Computer Science, Springer, Dortmund, p. 195, 1983.

[7] Gonzalez, T. F., and Zheng, S. Q., Bounds for partitioning rectilinear polygons, *Proc. ACM Symp. Computational Geometry,* 1985, p. 281.

[8] Du, D. Z., Pan, L. Q., and Shing, M. T., Minimum Edge Length Guillotine Rectangular Partition, Technical report, MSRI 02418–86, 1986.

[9] Gonzalez, T. F., and Zheng, S. Q., Improved bounds for rectangular and guillotine partitions, *J. Symbolic Comput.*, **7**, 591, 1989.

[10] Gonzalez, T. F., and Zheng, S. Q., Approximation algorithms for partitioning rectilinear polygons with interior points, *Algorithmica*, 5, 11, 1990.

[11] Gonzalez, T. F., Razzazi, M., and Zheng, S. Q., An efficient divide-and-conquer algorithm for partitioning into d-boxes", *Int. J. Comput. Geometry Appl.,* 3(4), 1993, 417 (condensed version appeared in *Proc. 2nd Canadian Conf. on Computational Geometry,* 1990, p. 214).

[12] Gonzalez, T. F., Razzazi, M., Shing, M., and Zheng, S. Q., On optimal *d*-guillotine partitions approximating hyperrectangular partitions, *Comput. Geometry: Theor. Appl.,* 4(1), 1, 1994.

[13] Kirkpatrick, D. G., An upper bound for sorting integers in restricted ranges, in *Proc. 18th Allerton Conf. on Communication, Control and Computing,* Monticello, Illinois, 1980.

[14] Cardei, M., Cheng, X., Cheng, X., and Du, D. Z., A tale on guillotine cut, in *Proc. Novel Approaches to Hard Discrete Optimization,* Ontario, Canada, 2001.

55

Partitioning Finite d-Dimensional Integer Grids with Applications

Silvia Ghilezan
University of Novi Sad

Jovanka Pantović
University of Novi Sad

Joviša Žunić[*]
University of Exeter

55.1 Introduction

In this chapter we will consider partitions of finite d-dimensional integer grids, that is, sets of the form $\{0, 1, \ldots, m-1\}^d$, by lines in two-dimensional space or by hyperplanes and hypersurfaces in an arbitrary dimension. Different aspects of the problem depending on m, d, and the type of hypersurfaces used have been widely studied in different areas of computer science and mathematics. In this chapter we will focus on problems arising in the areas of digital image processing (analysis) and neural networks. For brevity, related problems arising in other areas of computing (e.g., multivalued logic) and in pure mathematics areas (e.g., group theory) will not be analyzed.

Our work may be viewed as a technique for representing planar digital objects and threshold functions in near-optimal space. In this sense it is a heuristic for minimizing the space required to represent these objects.

The chapter is organized as follows. Section 55.2 begins with basic definitions and notations and ends with the basic mathematical result that will be used in the chapter. This result determines which discrete moments are enough for a unique characterization of discrete point sets from a certain family. The result is very useful because discrete moments are easy and fast to compute (summations and multiplications are needed only) and because the number of moments needed is relatively small. In Section 55.3 we discuss planar digital objects that arise in the area of digital image analysis. These objects are characterized by variables m (image size), and $d = 2$ (corresponds to a 2D image) or $d = 3$ (corresponds to a 3D image). When $m \to \infty$ the image has a very high resolution. Due to the recent technology development one is able to collect a huge number of images, and consequently, new applications are creating new problems related to our problems. One of the actual problems is to determine whether or not an object (or a kind of objects) appears on some images from a given database. Straightforward methods to solve this problem do not work satisfactorily for several reasons: The image collection and the object size (in terms of the number of pixels) slows down the methods considerably, the photos are

[*] J. Žunić is also with the Mathematical Institute, Serbian Academy of Arts and Sciences, Belgrade.

taken from different view points, etc. Section 55.3 gives an encoding scheme for digital objects (digital images of real objects) that essentially reduces the time necessary to solve related object matching problems. We show that $\mathcal{O}\left(h^2 \log(m+n)\right)$ bits are enough for a unique matching of curves from an $m \times n$ binary image if the number of mutual intersection between them is at most h. The parameter h is usually application dependent and it is known beforehand—for example: Since boundaries of machine-made products consist of second degree curve arcs, then $h = 4$ is a natural presumption for these objects.

Particular attention is given to the encoding by least squares fit lines (surfaces). The least square fit method is used to represent scattered data by lines (or curves) to reduce storage requirements or to enable a proper visualization. We establish that under certain conditions the representation of discrete data by least squares is information lossless. The results presented in Section 55.4 are of interest in the area of neural networks. While the performance of conventional algorithms is measured by the space and time complexity, in the area of neural networks performance is based on information theory and computational complexity. The performance of a neural network consisting of a given number of pairwise connected (connections are weighted) neurons with "off" and "on" states is strongly related to partitions of d-dimensional hypercube by a single hyperplane. In our terminology this corresponds to the situation where $m = 2$ while d varies (increases) to reach the required neural network capacity, that is, the ability to achieve the required number of stable states that, for example, correspond to data from the training set. The encoding of such a linear partition (states) by our encoding scheme requires $\mathcal{O}\left(d^2\right)$ bits that is an asymptotic minimum. Thus, it enables a fastest possible comparison between those partitions.

Our concluding remarks are given in Section 55.5.

55.2 Preliminaries

55.2.1 Definitions and Notation

The following definitions and notation will be used throughout this chapter.

If a set X consists of a finite number of points (elements), this number will be denoted by $\#X$, that is, $\#X$ is the cardinality of X.

$\mathbb{R}$ denotes the set of real numbers. $\mathbb{Z}$ is the set of integers. $\mathbb{N}_0$ is the set of nonnegative integers.

If $X \subseteq \mathbb{R}^d$ is a finite number point set and $(p_1, p_2, \ldots, p_d) \in \mathbb{N}_0^d$, then $(p_1, p_2, \ldots, p_d)$-*discrete moment* $\mu_{p_1,p_2,\ldots,p_d}(X)$ is defined as

$$\mu_{p_1,p_2,\ldots,p_d}(X) = \sum_{(x_1,x_2,\ldots,x_d)\in X} x_1^{p_1} \cdot x_2^{p_2} \cdot \ldots \cdot x_d^{p_d}$$

The order of the moment $\mu_{p_1,p_2,\ldots,p_d}$ is said to be $p_1 + p_2 + \cdots + p_d$. The formal equality $0^0 = 1$ is applied. A continuous function $f(x_1, x_2, \ldots, x_d)$ is said to *separate* the sets A and B if the sign of $f(x_1, x_2, \ldots, x_d)$ in the points of A differs from the sign of $f(x_1, x_2, \ldots, x_d)$ in the points of B.

Definition 55.1

A function $f(x_1, x_2, \ldots, x_d)$ separates the sets A and B if and only if either (i) *or* (ii) *holds*

(i) $A \subset \{(x_1, x_2, \ldots, x_d) \mid f(x_1, x_2, \ldots, x_d) > 0\}$ *and* $B \subset \{(x_1, x_2, \ldots, x_d) \mid f(x_1, x_2, \ldots, x_d) < 0\}$,

(ii) $A \subset \{(x_1, x_2, \ldots, x_d) \mid f(x_1, x_2, \ldots, x_d) < 0\}$ *and* $B \subset \{(x_1, x_2, \ldots, x_d) \mid f(x_1, x_2, \ldots, x_d) > 0\}$.

For a function $f(x_1, x_2, \ldots, x_d)$ with the domain A, $f^{-1}(c)$ will denote the set of points for which $f(x_1, x_2, \ldots, x_d) = c$, that is, $f^{-1}(c) = \{(x_1, x_2, \ldots, x_d) \in A \mid f(x_1, x_2, \ldots, x_d) = c\}$.

55.2.2 Necessary Mathematics

The following theorem is the basic mathematical tool used in this chapter. It yields information about the set of discrete moments that is sufficient for the unique characterization of discrete point sets from a given class.

Theorem 55.1

Let A and B be two finite subsets of $\mathbb{R}^d$ and let a finite set $\mathcal{P} \subset \mathbb{N}_0^d$ be fixed. Suppose that the set differences $A \setminus B$ and $B \setminus A$ can be separated by a function $f(x_1, x_2, \ldots, x_d)$ of the form

$$f(x_1, x_2, \ldots, x_d) \;=\; \sum_{(p_1, p_2, \ldots, p_d) \in \mathcal{P}} a_{p_1, p_2, \ldots, p_d} \cdot x_1^{p_1} \cdot x_2^{p_2} \cdot \ldots \cdot x_d^{p_d} \tag{55.1}$$

for some real numbers $a_{p_1, p_2, \ldots, p_d}$. Then,

$$A = B \quad \Leftrightarrow \quad \mu_{p_1, p_2, \ldots, p_d}(A) = \mu_{p_1, p_2, \ldots, p_d}(B) \;\; \text{for all} \;\; (p_1, p_2, \ldots, p_d) \in \mathcal{P}$$

Proof

$A = B$ implies that all the corresponding moments must be equal. To prove that the assumed moment equalities preserve $A = B$, we will show that the assumptions $\mu_{p_1, p_2, \ldots, p_d}(A) = \mu_{p_1, p_2, \ldots, p_d}(B)$ (for all $(p_1, p_2, \ldots, p_d) \in \mathcal{P}$) and $A \neq B$ lead to a contradiction.

Since $A \neq B$ we can assume that $A \setminus B$ is nonempty, otherwise we can start with the nonempty $B \setminus A$.

To simplify the notation, we write $\mathbf{x}$ instead of $(x_1, x_2, \ldots, x_d)$, and $\mathbf{p}$ instead of $(p_1, p_2, \ldots, p_d)$. Also, $\alpha_{p_1, p_2, \ldots, p_d} = \alpha_{\mathbf{p}}$, while $\mathbf{x}^{\mathbf{p}} = x_1^{p_1} \cdot x_2^{p_2} \cdot \cdots \cdot x_d^{p_d}$.

From our assumption there is a function

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_d) = \sum_{(p_1, p_2, \ldots, p_d) \in \mathcal{P}} a_{p_1, p_2, \ldots, p_d} \cdot x_1^{p_1} \cdot x_2^{p_2} \cdot \ldots \cdot x_d^{p_d} = \sum_{\mathbf{p} \in \mathcal{P}} \alpha_{\mathbf{p}} \cdot \mathbf{x}^{\mathbf{p}}$$

that separates $A \setminus B$ and $B \setminus A$ and, let us assume, satisfies the condition (i). Therefore we have the following derivation:

$$0 < \sum_{\mathbf{x} \in A \setminus B} f(\mathbf{x}) - \sum_{\mathbf{x} \in B \setminus A} f(\mathbf{x}) = \sum_{\mathbf{x} \in A \setminus B} \left(\sum_{\mathbf{p} \in \mathcal{P}} \alpha_{\mathbf{p}} \cdot \mathbf{x}^{\mathbf{p}} \right) - \sum_{\mathbf{x} \in B \setminus A} \left(\sum_{\mathbf{p} \in \mathcal{P}} \alpha_{\mathbf{p}} \cdot \mathbf{x}^{\mathbf{p}} \right)$$

$$= \sum_{\mathbf{p} \in \mathcal{P}} \left(\alpha_{\mathbf{p}} \cdot \sum_{\mathbf{x} \in A \setminus B} \mathbf{x}^{\mathbf{p}} \right) - \sum_{\mathbf{p} \in \mathcal{P}} \left(\alpha_{\mathbf{p}} \cdot \sum_{\mathbf{x} \in B \setminus A} \mathbf{x}^{\mathbf{p}} \right) = \sum_{\mathbf{p} \in \mathcal{P}} \alpha_{\mathbf{p}} \cdot \mu_{\mathbf{p}}(A \setminus B) - \sum_{\mathbf{p} \in \mathcal{P}} \alpha_{\mathbf{p}} \cdot \mu_{\mathbf{p}}(B \setminus A)$$

$$= \sum_{\mathbf{p} \in \mathcal{P}} \alpha_{\mathbf{p}} \cdot \big(\mu_{\mathbf{p}}(A) - \mu_{\mathbf{p}}(A \cap B) \big) \quad - \left(\sum_{\mathbf{p} \in \mathcal{P}} \alpha_{\mathbf{p}} \cdot (\mu_{\mathbf{p}}(B) - \mu_{\mathbf{p}}(B \cap A)) \right) = 0$$

The derived contradiction $0 < \sum_{\mathbf{x} \in A \setminus B} f(\mathbf{x}) - \sum_{\mathbf{x} \in B \setminus A} f(\mathbf{x}) \leq 0$ completes our proof. □

55.3 Planar Digital Objects

Digital objects are defined to be the result of subjecting real objects to a certain digitization process. A planar continuous curve γ with the equation $y = f(x)$ is usually digitized so that the closest digital points (points with integer coordinates, often referred to as pixels) below a given curve are taken (see Figure 55.1[a]). The set of digital points associated to the digitized curve γ is called a *digital curve* and it is defined by

$$C(\gamma) = \{(i, \lfloor f(i) \rfloor), \;\; i \;\; \text{is an integer} \;\}$$

($\lfloor k \rfloor$ denotes the greatest integer not larger than k).

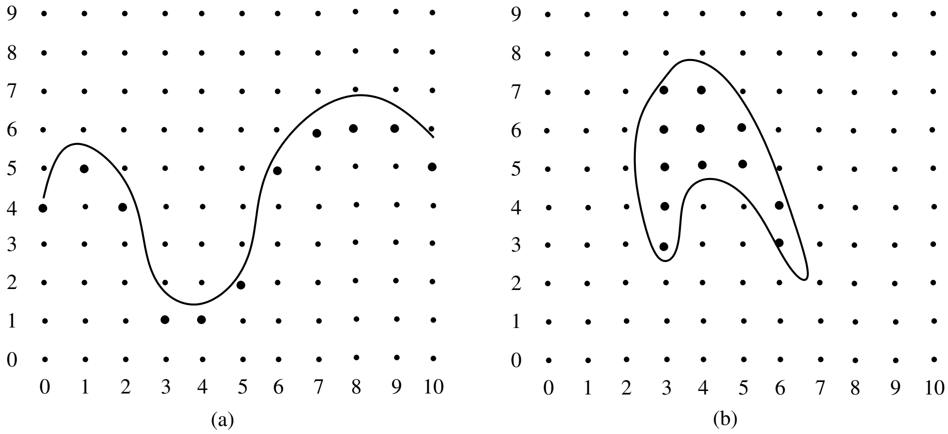


FIGURE 55.1 (a) Digitization of the given real curve on the 11 × 10 integer grid, consists of 11 enlarged points. (b) The digitization of the planar region (bounded by the presented curve) consists of 12 points inside it.

In practical applications, we deal with finite subsets of $C(\gamma)$ or, more precisely, with *digital curve segments* that are obtained by digitizing parts of curves lying between a pair of vertical lines $x = x_1$ and $x = x_2$. Without loss of generality, we can assume $x_1 = 0$ and $x_2 = m - 1$, where m is an integer.

We will be dealing with the digital curve segments $C_m(\gamma)$ having the form

$$C_m(\gamma) = \{(i, \lfloor f(i) \rfloor),\ i = 0, 1, 2, \ldots, m-1\} \tag{55.2}$$

Obviously, m is the number of digital points in the digital curve segment $C_m(\gamma)$. Naturally, if γ is a straight line then $C_m(\gamma)$ is called a digital straight line segment, if γ is part of a hyperbola, then $C_m(\gamma)$ is called a digital hyperbola segment, and so on.

Among the most important problems considered in digital image analysis are the recognition of original objects and the estimation of their relevant parameters (based on the data resulting from the digitization) as well as the creation of efficient coding schemes for digital objects. Such an efficient encoding scheme for digital objects should preserve low storage complexity, fast transmission, mutual comparison of digital objects, etc.

It is worth mentioning that if a planar curve is digitized, then the curve equation $y = f(x)$ and x-coordinates of the endpoints can be used for a trivial representation of its digitization. However, this trivial method has several deficiencies. As an illustration, we give two of them:

- In real applications, the equations of digitized curves are usually unknown. In dealing with satellite images, or images taken by a digital camera, for example, such equations are not provided. Moreover, even if we know that a binary image represents a real pyramid, whose edges are most likely straight line segments, we do not know the equations of such lines.
- There are infinitely many real curve segments (even of different kinds) whose digitizations coincide (as an example, it could happen that a hyperbola segment and a straight line segment have the same digitizations).

Therefore, it would be useful to have a one-to-one mapping between digital curve segments from a given domain and their representations. Note that the trivial method mentioned above does not satisfy this requirement.

55.3.1 A General Coding Scheme

In this subsection we show that a suitably chosen set of discrete moments can provide an efficient encoding of digital curve segments belonging to a given set. The number of required moments depends on the maximum number of intersection points between any two original curves whose digitization belongs to the set considered. We will show that digital curve segments (from a fixed set) can be coded by $h + 1$ discrete moments if there are no more than h intersection points between any pair of real curves whose digitizations are considered.

We start with a precise definition of a family Γ_h (h is an arbitrary integer) of sets of digital curve segments to which the coding scheme can be applied.

Definition 55.2

A set $\mathcal{G}$ of digital curve segments belongs to a family Γ_h if the following conditions hold:

(C1) $\mathcal{G}$ consists of digital curve segments which are digitizations of graphs of continuous, explicitly given functions $y = f(x) > 0$, where $x \in [0, m-1]$.

(C2) Let $\mathbf{G}_1,\ \mathbf{G}_2 \in \mathcal{G}$. Then there are two continuous curves γ_1 and γ_2 such that $\mathbf{G}_1 = C_m(\gamma_1)$ and $\mathbf{G}_2 = C_m(\gamma_2)$ and γ_1 and γ_2 have no more than h intersection points on the interval $[0, m-1]$.

The sets which are the most interesting for practical reasons are the set of digital straight line segments (from the family Γ_1), the set of subarcs of digital half-circle segments (from the family Γ_2), the set of subarcs of digital half-ellipse segments (from the family Γ_4), the set of digital cubic parabola segments (the family Γ_3), as well as unions of these sets. Particular attention will be given to the sets of digital polynomial segments having a degree not exceeding a given number r (from the family Γ_r).

The following theorem shows that all digital curve segments belonging to a fixed $\mathcal{G} \in \Gamma_h$ can be matched uniquely by $h + 1$ discrete moments.

Theorem 55.2

Let a set $\mathcal{G}$ from the family Γ_h be given and let two digital curve segments $\mathbf{G}_1 \in \mathcal{G}$ and $\mathbf{G}_2 \in \mathcal{G}$. Then

$$\mathbf{G}_1 = \mathbf{G}_2 \quad \Leftrightarrow \quad \mu_{0,1}(\mathbf{G}_1) = \mu_{0,1}(\mathbf{G}_2),\ \ \mu_{1,1}(\mathbf{G}_1) = \mu_{1,1}(\mathbf{G}_2),\ \ \ldots,\ \ \mu_{h,1}(\mathbf{G}_1) = \mu_{h,1}(\mathbf{G}_2)$$

Proof

We will show that the statement is a consequence of Theorem 55.1. For a given curve γ (where γ is the graph of a function $y = f(x) > 0$ on $[0, m-1]$) let $\mathbf{R}(\gamma)$ be the set of all integer points from the closed area bounded by γ and by the lines: $x = 0$, $y = 0$, and $x = m - 1$ (see Figure 55.2 [a]).

It is obvious that

$$C_m(\gamma_1) = C_m(\gamma_2) \quad \Leftrightarrow \quad \mathbf{R}(\gamma_1) = \mathbf{R}(\gamma_2) \tag{55.3}$$

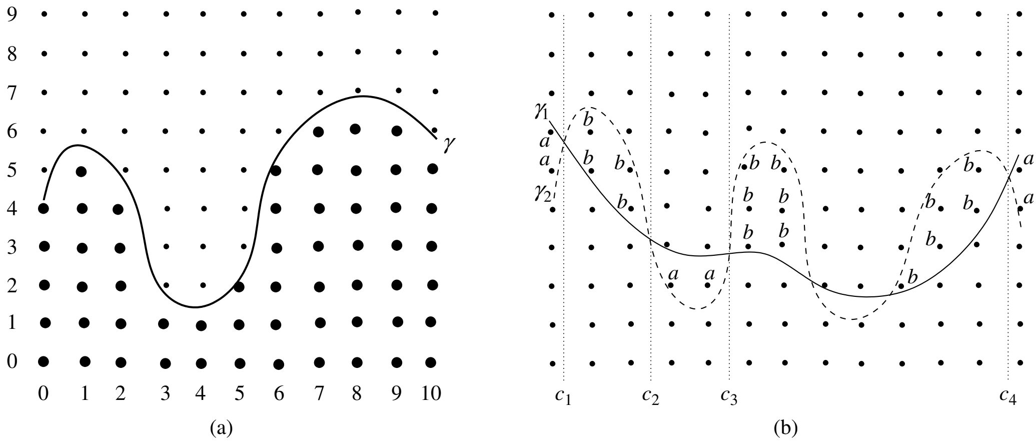
since by Definition 55.2 there are two continuous curves γ_1 and γ_2 such that $\mathbf{G}_1 = C_m(\gamma_1)$ and $\mathbf{G}_2 = C_m(\gamma_2)$ and γ_1 and γ_2 have no more than h intersection points on the interval $[0, m-1]$.

Since $\mu_{k,1}(C_m(\gamma)) = \sum\limits_{(i,j)\in C_m(\gamma)} i^k \cdot \lfloor f(i) \rfloor$ equals the sum of kth powers of the abscissa values of all digital points belonging to $\mathbf{R}(\gamma)$ except those belonging to the x-axis, we have the following equalities.

$$\begin{aligned}
\mu_{0,1}(C_m(\gamma)) &= \sum_{(i,j)\in C_m(\gamma)} \lfloor f(i) \rfloor &&= \sum_{(i,j)\in \mathbf{R}(\gamma)} 1 - \sum_{(i,0)\in \mathbf{R}(\gamma)} 1 &&= \mu_{0,0}(\mathbf{R}(\gamma)) - s_0\\
\mu_{1,1}(C_m(\gamma)) &= \sum_{(i,j)\in C_m(\gamma)} i \cdot \lfloor f(i) \rfloor &&= \sum_{(i,j)\in \mathbf{R}(\gamma)} i - \sum_{(i,0)\in \mathbf{R}(\gamma)} i &&= \mu_{1,0}(\mathbf{R}(\gamma)) - s_1\\
&\ \ \cdots && \quad\cdots && \quad\cdots\\
\mu_{h,1}(C_m(\gamma)) &= \sum_{(i,j)\in C_m(\gamma)} i^h \cdot \lfloor f(i) \rfloor &&= \sum_{(i,j)\in \mathbf{R}(\gamma)} i^h - \sum_{(i,0)\in \mathbf{R}(\gamma)} i^h &&= \mu_{h,0}(\mathbf{R}(\gamma)) - s_h,
\end{aligned} \tag{55.4}$$

where $s_0 = \sum\limits_{(i,0)\in \mathbf{R}(\gamma)} 1 = m$, $s_1 = \sum\limits_{(i,0)\in \mathbf{R}(\gamma)} i = \frac{m\cdot(m-1)}{2}, \ldots, s_h = \sum\limits_{(i,0)\in \mathbf{R}(\gamma)} i^h$ are the constants which depend on m but not on γ. Let us consider the set differences $\mathbf{R}(\gamma_1)\setminus\mathbf{R}(\gamma_2)$ and $\mathbf{R}(\gamma_2)\setminus\mathbf{R}(\gamma_1)$. By definition:

- $\mathbf{R}(\gamma_1) \setminus \mathbf{R}(\gamma_2)$ consists of integer points lying below the curve γ_1 and above the curve γ_2, while $\mathbf{R}(\gamma_2) \setminus \mathbf{R}(\gamma_1)$ consists of integer points lying below the curve γ_2 and above the curve γ_1 (see Figure 55.2[b]).
- γ_1 and γ_2 have no more than h intersection points on the interval $[0, m-1]$.



(a)

(b)

FIGURE 55.2 (a) The set of integer points $\mathbf{R}(\gamma)$ corresponding to the displayed curve γ consists of all the enlarged points. (b) The integer points belonging to $\mathbf{R}(\gamma_1) \setminus \mathbf{R}(\gamma_2)$ and to $\mathbf{R}(\gamma_2) \setminus \mathbf{R}(\gamma_1)$ are labeled by a and b, respectively.

Let γ_1 and γ_2 be the graphs of continuous functions $y = f_1(x)$ and $y = f_2(x)$, respectively, and let $c_1, c_2, \ldots, c_l$ (with $l \leq h$) be all the intersection points between γ_1 and γ_2 on $[0, m-1]$.

It is easy to see that there are parallel strips

$$-\infty < x < c_{i_1}, \quad c_{i_1} < x < c_{i_2}, \ \ldots, c_{1_{s-1}} < x < c_{i_s}, \quad c_{i_s} < x < \infty$$

such that

- $\{c_{i_1}, c_{i_2}, \ldots, c_{i_s}\} \subset \{c_1, c_2, \ldots, c_l\}$.
- any strip contains at least one integer point either from $\mathbf{R}(\gamma_1) \setminus \mathbf{R}(\gamma_2)$ or from $\mathbf{R}(\gamma_2) \setminus \mathbf{R}(\gamma_1)$.
- there is no strip containing points from both $\mathbf{R}(\gamma_1) \setminus \mathbf{R}(\gamma_2)$ and $\mathbf{R}(\gamma_2) \setminus \mathbf{R}(\gamma_1)$.
- if a strip contains points from $\mathbf{R}(\gamma_1) \setminus \mathbf{R}(\gamma_2)$ then its neighbouring strip contains integer points from $\mathbf{R}(\gamma_2) \setminus \mathbf{R}(\gamma_1)$ (not any point from $\mathbf{R}(\gamma_1) \setminus \mathbf{R}(\gamma_2)$).

Under these assumptions, the function

$$f(x) = (x - c_{i_1}) \cdot (x - c_{i_2}) \cdot \cdots \cdot (x - c_{i_s})$$

divides the plane in two parts: $f(x) > 0$ and $f(x) < 0$ (either part consists of alternately taken parallel strips defined above) in such a way that $\mathbf{R}(\gamma_1) \setminus \mathbf{R}(\gamma_2)$ belongs to one part (let us say to the part where $f(x) > 0$) while $\mathbf{R}(\gamma_2) \setminus \mathbf{R}(\gamma_1)$ belongs to the other part (i.e., to the part with $f(x) < 0$). In other words, the function $f(x) = (x - c_{i_1}) \cdot (x - c_{i_2}) \cdot \cdots \cdot (x - c_{i_s})$ separates the set differences $\mathbf{R}(\gamma_1) \setminus \mathbf{R}(\gamma_2)$ and $\mathbf{R}(\gamma_2) \setminus \mathbf{R}(\gamma_1)$. Consequently, in accordance with Theorem 55.1,

$$\mathbf{R}(\gamma_1) = \mathbf{R}(\gamma_2) \quad \Leftrightarrow \quad \mu_{i,0}(\mathbf{R}(\gamma_1)) = \mu_{i,0}(\mathbf{R}(\gamma_2)) \quad \text{for} \ \ i = 0, 1, \ldots, l \tag{55.5}$$

Since $s < h$ and because of Eq. (55.4) and Eq. (55.5), the statement is proven. □

The next corollary is a straightforward consequence of the previous theorem.

Corollary 55.1

The set $\mathcal{P}_r$ of digital polynomial segments with a degree less or equal to r can be coded uniquely by $r + 1$ discrete moments.

The proposed coding scheme can be understood as very efficient with respect to its memory space requirements. To be more precise, if a set $\mathcal{G} \in \Gamma_h$ is fixed, and if all digital curve segments from $\mathcal{G}$ could be presented on an $m \times n$ integer grid, then

$$\mu_{k,1}(C_m(\gamma)) = \sum_{(i,j)\in C_m(\gamma)} j \cdot i^k = \mathcal{O}\left(n \cdot m^{k+1}\right)$$

follows easily for any k. The last asymptotic estimate applied to the moments $\mu_{0,1}(C_m(\gamma))$, $\mu_{1,1}(C_m(\gamma))$, $\ldots$, $\mu_{h,1}(C_m(\gamma))$ leads to the following theorem that describes the storage complexity of the proposed encoding scheme.

Theorem 55.3

Fix $\mathcal{G} \in \Gamma_h$. If all digital curve segments from $\mathcal{G}$ are presented on $\{0, 1, \ldots, m-1\} \times \{0, 1, \ldots n-1\}$, then the encoding suggested by Theorem 55.2 requires

$$\mathcal{O}\left(h^2 \cdot \log(n+m)\right)$$

bits per coded digital curve segment from $\mathcal{G}$.

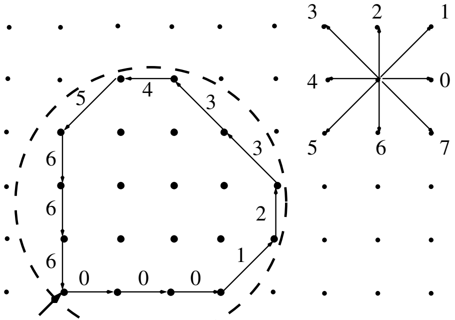
Observation 55.1

In the case when h is assumed to be a constant, $\mathcal{O}(\log(m+n))$ bits are sufficient, which is obviously the theoretical minimum. For example, if $\mathcal{G}$ consists of digital straight line segments (consisting of m points) inscribed into an $n \times n$ integer grid, then $\#\mathcal{G} = \frac{3 \cdot n^4}{\pi^2} + \mathcal{O}\left(n^3 \cdot \log n\right)$ (see Refs. [1,2]) showing that the previous bit rate is the theoretical minimum (taken in an asymptotic sense).

To illustrate the efficiency of the above coding scheme we cite an example from Ref. [3]. The comparison of the code proposed here and the well-known Freeman code [4] (also called the 8-chain code—see Figure 55.3) for the digitization of the curve γ is made under the assumption that γ belongs to a set from Γ_3. Under such an assumption, the moment based code of $C_{82}(\gamma)$ is

$$(\mu_{0,1}(C_{82}(\gamma)), \mu_{1,1}(C_{82}(\gamma)), \mu_{2,1}(C_{82}(\gamma)), \mu_{3,1}(C_{82}(\gamma))) = (5030,\ 215522,\ 12020658,\ 736586012)$$

while the 8-chain code (represented by a sequence of digits (0–7) indicating the direction of the next point [as given in Figure 55.3]) is 7 6 7 7 6 7 6 7 6 7 7 6 7 6 7 6 7 7 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 1 2 1 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 0 0 0 7 0 0 0 0 0 7 0 0 0 0 1 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 7 7 6 7 7 6 7 7 7 6 7 7 6 1 1 2 2 1 2 2 2 1 2 2 1 2 2 2 1 2 2 1 2 2 2 1 2 7 6 6 7 6 6 6 7 6 6 1 2 1 2 2 2 1 2 2 1 2 2 2 7 6 6 6 6 6 6 7 6 6 6 6 6 6 6 7 6 6 1 2 2 1 2 1 2 2 1 2 2 7 7 6 6 6 6 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 6.



Freeman code is 0 0 0 1 2 3 3 4 5 6 6 6

FIGURE 55.3 The boundary of the presented digital disc (enlarged points) is coded by the well-known Freeman code.

The previous example is just one illustration that coincides with theoretical observations. Precisely, if a curve having the length l is presented on binary image then the storage of all digital points belonging to such a binary image requires $\mathcal{O}(l \log l)$ bits and this bound, in general, cannot be improved. The encoding by the 8-chain code requires $\mathcal{O}(l)$ bits, while the complexity of our encoding scheme depends on the parameter h and for $h = \mathcal{O}(l^{\frac{1}{2}-\varepsilon})$ (where $\varepsilon > 0$) it is dominant. The last condition is not a strong limitation for real applications.

55.3.2 Encoding by Least Squares Fit Polynomials

Least squares fit lines (or more generally fit polynomials) are commonly used in data visualization and data representation. They are easy to compute and it is also expected that they look like the presented set of points.

It is clear that least squares fit lines (polynomials) corresponding to different discrete point sets could coincide if there are no restrictions to the discrete point sets fitted. But it could be of interest to characterize the situations when they are necessarily different, that is, when the mapping

$$\text{discrete point set} \quad \longrightarrow \quad \text{corresponding least squares fit polynomial}$$

is one to one. If a set $\mathcal{D}$ of digital objects has such a nice property then any digital object from $\mathcal{D}$ can be represented uniquely by the corresponding least squares fit polynomial, that is, by its coefficients. In this case a comparison between elements of $\mathcal{D}$ could be made by comparing the corresponding least squares fit polynomials.

The following question was proposed in Ref. [5]: *Does a least squares fit line uniquely match a digital straight line segment on a given interval?* A positive answer is given in Ref. [6]. Later on, the result is generalized [3] to least squares fit polynomials. Moreover, it turns out that the encoding by least squares fit polynomial is a subcase of the proposed general coding scheme.

Let S be a finite discrete point set. The polynomial of a given degree that minimizes the total sum of the squares of vertical distances from the polynomial to the points from S can be determined easily. Indeed, if the equation of the required polynomial is $p(x) = a_r x^r + a_{r-1}x^{r-1} + \cdots + a_0$, then the function

$$F(a_r, a_{r-1}, \ldots, a_0) = \sum_{(x,y)\in S} (p(x) - y)^2 = \sum_{(x,y)\in S} (a_r x^r + a_{r-1}x^{r-1} + \cdots + a_0 - y)^2$$

should be minimized. Thus the following system of $r + 1$ equations has to be satisfied:

$$\frac{\partial F}{\partial a_r} = 0, \qquad \frac{\partial F}{\partial a_{r-1}} = 0, \qquad \ldots, \qquad \frac{\partial F}{\partial a_0} = 0$$

If $\mathbf{S}$ is a digital curve segment $\mathbf{S} = \{(0, y_0), (1, y_1), \ldots, (m-1, y_{m-1})\}$ then the above system becomes

$$\begin{aligned} S_{2r} \cdot a_r + S_{2r-1} \cdot a_{r-1} + \cdots + S_r \cdot a_0 &= \sum_{i=0}^{m-1} y_i i^r (= \mu_{r,1}(\mathbf{S})) \\ S_{2r-1} \cdot a_r + S_{2r-2} \cdot a_{r-1} + \cdots + S_{r-1} \cdot a_0 &= \sum_{i=0}^{m-1} y_i i^{r-1} (= \mu_{r-1,1}(\mathbf{S})) \\ &\cdots \\ S_r \cdot a_r + S_{r-1} \cdot a_{r-1} + \cdots + S_0 \cdot a_0 &= \sum_{i=0}^{m-1} y_i (= \mu_{0,1}(\mathbf{S})) \end{aligned} \tag{55.6}$$

where the coefficients $S_0, S_1, \ldots, S_{2r}$ are given by

$$S_j = \sum_{i=0}^{m-1} i^j, \quad 0 \leq j \leq 2r$$

The unknown $a_r, a_{r-1}, \ldots, a_0$ are the coefficients of the least squares fit polynomial. If $m \geq r$, the determinant of the system (55.6) is different from zero (see Ref. [3]) and the system has a unique solution. Trivially, for $m < r$ there are infinitely many solutions.

Let $\mathbf{S} = C_m(\gamma) \in \mathcal{G} \in \Gamma_r$ and let $a_r(\gamma), a_{r-1}(\gamma), \ldots, a_0(\gamma)$ be the solution of (55.6). A very practical question is:

> *Are there two different digital curve segments, $C_m(\gamma_1)$ and $C_m(\gamma_2)$, both from a given set $\mathcal{G} \in \Gamma_r$, such that $a_r(\gamma_1) = a_r(\gamma_2), a_{r-1}(\gamma_1) = a_{r-1}(\gamma_2), \ldots, a_0(\gamma_1) = a_0(\gamma_2)$?*

The answer is negative. This means that digital curve segments from a fixed set belonging to Γ_r and their least squares fit polynomials having the degree r are in a one-to-one correspondence. This enables the encoding of the digital curve segments by their associated least squares polynomials. This is stated by the following theorem.

Theorem 55.4

Let $C_m(\gamma_1)$ and $C_m(\gamma_2)$ be two digital curve segments from a set belonging to the family Γ_r. If $a_r(\gamma_1)$, $a_{r-1}(\gamma_1), \ldots, a_0(\gamma_1)$ and $a_r(\gamma_2), a_{r-1}(\gamma_2), \ldots, a_0(\gamma_2)$ are the coefficients of the least squares fit polynomials associated to $C_m(\gamma_1)$ and $C_m(\gamma_2)$, respectively, then

$$a_r(\gamma_1) = a_r(\gamma_2) \;\; \textit{and} \;\; a_{r-1}(\gamma_1) = a_{r-1}(\gamma_2) \;\; \textit{and} \ldots \textit{and} \;\; a_0(\gamma_1) = a_0(\gamma_2) \;\; \Leftrightarrow \;\; C_m(\gamma_1) = C_m(\gamma_2)$$

Proof

Trivially, $C_m(\gamma_1) = C_m(\gamma_2) \Longrightarrow a_i(\gamma_1) = a_i(\gamma_2)$ for all $0 \le i \le r$.

We prove that the opposite direction is a consequence of Theorem 55.1.

Let $\mathbf{M}$ denote the matrix of the system (55.6), that is, $\mathbf{M} = \begin{bmatrix} S_{2r} & S_{2r-1} & \ldots & S_r \\ S_{2r-1} & S_{2r-2} & \ldots & S_{r-1} \\ \ldots & \ldots & \ldots & \\ S_r & S_{r-1} & \ldots & S_0 \end{bmatrix}$

Then Eq. (55.6) gives

$$\mathbf{M} \cdot [a_r(\gamma_1),\;\; a_{r-1}(\gamma_1), \ldots, a_0(\gamma_1)]^T \;\; = \;\; [\mu_{r,1}(C_m(\gamma_1)),\;\; \mu_{r-1,1}(C_m(\gamma_1)), \ldots, \mu_{0,1}(C_m(\gamma_1))]^T$$

and

$$\mathbf{M} \cdot [a_r(\gamma_2),\; a_{r-1}(\gamma_2),\; \ldots,\; a_0(\gamma_2)]^T \; = \; [\mu_{r,1}(C_m(\gamma_2)),\;\; \mu_{r-1,1}(C_m(\gamma_2)),\; \ldots,\; \mu_{0,1}(C_m(\gamma_2))]^T$$

Since the corresponding least squares fit polynomials are equal, that is,

$$[a_r(\gamma_1), a_{r-1}(\gamma_1), \ldots, a_0(\gamma_1)] = [a_r(\gamma_2), a_{r-1}(\gamma_2), \ldots, a_0(\gamma_2)]$$

we obtain

$$\begin{aligned} &[\mu_{r,1}(C_m(\gamma_1)), \mu_{r-1,1}(C_m(\gamma_1)), \ldots, \mu_{0,1}(C_m(\gamma_1))] \\ &= [\mu_{r,1}(C_m(\gamma_2)), \mu_{r-1,1}(C_m(\gamma_2)), \ldots, \mu_{0,1}(C_m(\gamma_2))] \end{aligned}$$

The last equality together with Theorem 55.1 completes the proof. □

A direct consequence of the previous theorem is the following corollary.

Corollary 55.2

Let $P_m(\gamma)$ and $P_m(\beta)$ be two digital polynomial segments, where γ and β are polynomials with a degree less or equal to r. If $a_r(\gamma), a_{r-1}(\gamma), \ldots, a_0(\gamma)$ and $a_r(\beta), a_{r-1}(\beta), \ldots, a_0(\beta)$ are the coefficients of the least squares fit polynomials associated to $P_m(\gamma)$ and $P_m(\beta)$, respectively then:

$$a_r(\gamma) = a_r(\beta) \quad \textit{and} \quad a_{r-1}(\gamma) = a_{r-1}(\beta) \; \textit{and} \ldots \textit{and} \; a_0(\gamma) = a_0(\beta) \Longleftrightarrow P_m(\gamma) = P_m(\beta)$$

To close this subsection, let us mention that the encoding by the least squares fit objects (in general) does not preserve an optimal encoding. The coefficients are not necessarily integers. If they are stored as fractions, they could be extremely big since denominators and nominators are computed by using the system (55.6).

55.3.3 Encoding Planar Regions

Digital (binary) images of planar regions consist of pixels whose centers belong to the digitized region (see Figure 55.1[b]). Let a planar region R be bounded by a simple closed curve $f(x, y) = \sum_{(p_1, p_2) \in \mathcal{P}} a_{p_1, p_2} \cdot x^{p_1} \cdot x^{p_2}$ for some chosen set $\mathcal{P}$ and some chosen real number a_{p_1, p_2}, where $(p_1, p_2) \in \mathcal{P}$. A straightforward consequence of Theorem 55.1 is that the digitization of R, that is, $\mathbf{R} = R \cap \mathbb{Z}^2$ is uniquely represented by the moments $\mu_{p_1, p_2}(\mathbf{R})$, where (p_1, p_2) takes all the values from $\mathcal{P}$. Namely, the function $f(x, y) = \sum_{(p_1, p_2) \in \mathcal{P}} a_{p_1, p_2} \cdot x^{p_1} \cdot x^{p_2}$ separates the set differences $\mathbf{R} \setminus A$ and $A \setminus \mathbf{R}$ for any chosen discrete point set A.

The next theorem illustrates that the set of necessary moments sometimes can be reduced. It will be shown that any digital disc $\mathbf{D} = D \cap \mathbb{Z}^2$, defined as the intersection of a real disc D and squared integer grid $\mathbb{Z}^2$ could be matched uniquely by $\mu_{0,0}(\mathbf{D})$, $\mu_{1,0}(\mathbf{D})$, and $\mu_{0,1}(\mathbf{D})$. In other words, even if the disc boundary is a second degree polynomial, the first order moments are sufficient for the encoding.

Theorem 55.5

Let $\mathbf{D}_1$ and $\mathbf{D}_2$ be two digital discs. Then

$$\mathbf{D}_1 = \mathbf{D}_2 \quad \Leftrightarrow \quad \mu_{0,0}(\mathbf{D}_1) = \mu_{0,0}(\mathbf{D}_1) \ \textit{and}\ \mu_{1,0}(\mathbf{D}_1) = \mu_{1,0}(\mathbf{D}_2) \ \textit{and}\ \mu_{0,1}(\mathbf{D}_1) = \mu_{0,1}(\mathbf{D}_2).$$

Proof

Let $\mathbf{D}_1 = D_1 \cap \mathbb{Z}^2$ and $\mathbf{D}_2 = D_2 \cap \mathbb{Z}^2$, where D_1 and D_2 are real discs. To prove the statement, it is enough to see that $\mathbf{D}_1 \setminus \mathbf{D}_2$ and $\mathbf{D}_2 \setminus \mathbf{D}_1$ could be separated by a line of the form: $a \cdot x + b \cdot y + c = 0$. Precisely, if D_1 and D_2 intersect then the line determined by their intersection points can be used, else, a common tangent to D_1 and D_2 separates $\mathbf{D}_1$ and $\mathbf{D}_2$. The statement follows from Theorem 55.1. □

As mentioned before, in real applications, the original objects remain unknown during the digitization process. Thus, if we have a digital image of a real disc, we do not know the equation of the original circle that bounds the disc considered. In any case, even if we do not have the equation of the original circle (disc) it could be useful to have a circle (disc) that fits well with the resulting digital data. It is worth mentioning that the least squares fitting circle cannot be computed—only a numerical (approximative) solution is possible. The most common approach to computing a real circle (disc) that fits well with the considered digital disc $\mathbf{D}$ is to take the disc $\{(x, y) \mid (x - a)^2 + (x - b)^2 \leq r^2\}$, where the parameters a, b, and r are computed as follows:

$$a = \frac{\mu_{1,0}(\mathbf{D})}{\mu_{0,0}(\mathbf{D})}, \quad b = \frac{\mu_{0,1}(\mathbf{D})}{\mu_{0,0}(\mathbf{D})}, \quad r = \sqrt{\frac{1}{\pi} \cdot \mu_{0,0}(\mathbf{D})}$$

It can be shown that the disc constructed as described above is a very good approximation of the real circle, particularly if the reconstruction is based on a digital image with a very high resolution. For the limitation in estimating the real moments from the corresponding binary images we refer to Ref. [7].

To close this section, let us compare encoding by the Freeman code and encoding by moments $\mu_{0,0}$, $\mu_{1,0}$, and $\mu_{0,1}$ if both are applied to digital discs. The situation when the unit disc $U = (x - a)^2 + (x - b)^2 \leq 1$ (in an arbitrary position with respect to digitization grid) is presented on a digital image having the resolution ρ (i.e., there are ρ pixels per measure unit) corresponds to the situation when the disc $\rho \cdot U = (x - \rho \cdot a)^2 + (x - \rho \cdot b)^2 \leq \rho^2$ (i.e., the disc U dilated by the factor ρ) is digitized on the integer grid. Hence, if we consider the digitization of $\rho \cdot U$ on the integer grid, it is clear that the Freeman code requires $\mathcal{O}(\rho)$ bits (see Figure 55.3)—more precisely, the number of bits required has the same order as the perimeter of the disc. However, $\mathcal{O}(\log \rho)$ bits are sufficient if $\rho \cdot U$ is digitized on $\{0, 1, 2, \ldots, \rho, \rho + 1\} \times \{0, 1, 2, \ldots, \rho, \rho + 1\}$. Indeed, since $\mu_{0,0}(\rho \cdot U \cap \mathbb{Z}^2) = \mathcal{O}(\rho^2)$, $\mu_{1,0}(\rho \cdot U \cap \mathbb{Z}^2) = \mathcal{O}(\rho^3)$, and $\mu_{0,1}(\rho \cdot U \cap \mathbb{Z}^2) = \mathcal{O}(\rho^3)$ a number of $\mathcal{O}(\log \rho)$ bits is sufficient for the storage of $\mu_{0,0}(\rho \cdot U)$, $\mu_{1,0}(\rho \cdot U)$, and $\mu_{0,1}(\rho \cdot U)$.

55.3.4 On 3D Digital Objects

The digitization of a 3D curve is not well defined, for example, it is difficult to say what the most natural definition of a 3D digital line segment or 3D digital circle is. The situation is much more clear when we work with surfaces and bodies. Analogous to 2D, we can say that a surface is digitized when the closest points below the surface are taken. If we consider the digitization of a surface segment α on an rectangular domain, say $[0, m-1] \times [0, n-1]$ in the xy-plane, then the digital surface segment obtained can be represented as

$$S_{m,n}(\alpha) = \{\ (i,\ j,\ \lfloor f(i,\ j)\rfloor)\ \mid\ (i,\ j) \in \{0, 1, \ldots, m-1\} \times \{0, 1, \ldots, n-1\}\ \}$$

where α is the graph of the function $z = f(x, y)$. If $f(x, y) = a \cdot x + b \cdot y + c \cdot z + d$, then we have a digital plane segment. By using the same technique as in the 2D case, it can be shown that the set of digital plane segments defined on the same domain could be coded uniquely by the following moments: $\mu_{0,0,1}$, $\mu_{1,0,1}$, and $\mu_{0,1,1}$. What is more important is that the digital plane segments and their corresponding least squares fit planes are in a one-to-one correspondence. For more details, we refer to Ref. [8].

Digitization of three-dimensional bodies consists of all the integer points belonging to the digitized body. Similarly, as in the case of planar discs, it can be shown that the set of digital balls in 3D can be coded by four corresponding moments having the order up to 1. Readers interested in additional details are referred to Ref. [9].

55.4 Threshold Functions on Binary Inputs

In this section we will demonstrate how Theorem 55.1 can be applied to problems related to encoding and enumerating threshold functions. Generally speaking, a function $f(x_1, x_2, \ldots, x_d)$ is a threshold function if

$$f(x_1, x_2, \ldots, x_d) = sgn(F(x_1, x_2, \ldots, x_d)) = \begin{cases} 1 & \text{if} \quad F(x_1, x_2, \ldots, x_d) > 0 \\ 0 & \text{if} \quad F(x_1, x_2, \ldots, x_d) < 0 \end{cases} \tag{55.7}$$

for some real function $F(x_1, x_2, \ldots, x_d)$. This subject has a history of more than 40 years [10,11], but there are many research problems still open. That is not only caused by the diversity of the threshold functions studied, but also by difficulties in solving some of them. Threshold functions are usually large objects—for example, a single threshold function $f(x_1, x_2, \ldots, x_d)\ :\quad \{0, 1\}^d \rightarrow \{0, 1\}$ if defined as a vector in $\{0, 1\}^{2^d}$ space [12] requires 2^d bits for storage. Therefore, it would be useful to have an efficient coding scheme that requires a relatively small amount of bits per coded function. Such a coding scheme would enable a comparison of threshold functions without comparing their values at all points from their domain.

The enumeration problem is also interesting. The number of threshold functions from some class describes the information capacity of the class [13,14]. Encoding and enumeration problems are usually considered together and very often the number of threshold functions of a given kind is estimated from the size of the corresponding codes.

A function $f(x_1, x_2, \ldots, x_d)$ is a linear threshold function if there are real numbers $a_1, a_2, \ldots, a_d$, and t such that

$$f(x_1, x_2, \ldots, x_d) = \begin{cases} 1 & \text{if} \quad a_1x_1 + a_2x_2 + \cdots + a_dx_d + t > 0 \\ 0 & \text{if} \quad a_1x_1 + a_2x_2 + \cdots + a_dx_d + t < 0 \end{cases}$$

Linear threshold functions defined on $\{0, 1\}^d$ are studied mostly because of their importance in the theory of neural networks [12,15]. The set of linear threshold functions defined on $\{0, 1\}^d$ will be denoted by **LT**. The following simple and efficient characterization of linear threshold functions defined on $\{0, 1\}^d$ has been established by Chow [10,11].

Theorem 55.6

Let $f(x_1, x_2, \ldots, x_d)$ *and* $g(x_1, x_2, \ldots, x_d)$ *map* $\{0, 1\}^d \to \{0, 1\}$. *Also, let*

$$\mu_{p_1, p_2, \ldots, p_d}(f^{-1}(1)) = \mu_{p_1, p_2, \ldots, p_d}(g^{-1}(1))$$

for all $p_1, p_2, \ldots, p_d$ *such that* $0 \le p_1 + p_2 + \ldots + p_d \le 1$. *Then*

$$f(x_1, x_2, \ldots, x_d) \in \mathbf{LT} \quad \Rightarrow \quad g(x_1, x_2, \ldots, x_d) = f(x_1, x_2, \ldots, x_d)$$

In other words, if a linear threshold function $f(x_1, x_2, \ldots, x_d)$ is represented by $d + 1$ integers which are discrete moments (having the order up to 1) corresponding to $f^{-1}(1)$, then there is no other threshold function (even nonlinear) with the same representation. In the literature, such moments (or their slight modifications [16]) are known as *Chow parameters.* Based on the Chow parameters, the total number of linear threshold functions defined on $\{0, 1\}^d$ has been upper bounded as

$$\#\mathbf{LT} \le 2 \cdot 2^{d^2} \tag{55.8}$$

It is worth mentioning here that a lower bound for #**LT** has been proven recently [17]. More precisely, it has been shown that the inequality

$$\log(\#\mathbf{LT}_1) \ge d^2 \cdot \left(1 - \frac{10}{\log d}\right) \tag{55.9}$$

holds for a sufficiently large d.

Now, we show that Theorem 55.6 is a simple consequence of Theorem 55.1.

Let $f(x_1, x_2, \ldots, x_d) = sgn(a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_d \cdot x_d + t)$ be a linear threshold function and $g(x_1, x_2, \ldots, x_d) : \{0, 1\}^d \to \{0, 1\}$ be another arbitrary (not necessarily linear threshold) function and let the Chow parameters corresponding to $f(x_1, x_2, \ldots, x_d)$ and $g(x_1, x_2, \ldots, x_d)$ coincide. It is obvious that $a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_d \cdot x_d + t$ separates $f^{-1}(1) \setminus B$ and $B \setminus f^{-1}(1)$ for any $B \subset \{0, 1\}^d$. By setting $B = g^{-1}(1)$ and noticing that $a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_d \cdot x_d + t$ is of the form Eq. (55.1) with $\mathcal{P} = \{(p_1, p_2, \ldots, p_d) \mid 0 \le p_1 + p_2 + \cdots + p_d \le 1\} \subset \{0, 1\}^d$ we get that the equality $f^{-1}(1) = g^{-1}(1)$ holds due to Theorem 55.1, or equivalently $f(x_1, x_2, \ldots, x_d) = g(x_1, x_2, \ldots, x_d)$.

Theorem 55.1 leads to some improvements of the Chow result (for details see Ref. [18]):

- It enables us to show that two linear threshold functions $f(x_1, x_2, \ldots, x_d)$ and $g(x_1, x_2, \ldots, x_d)$ which satisfy $f(0, 0, \ldots, 0) = g(0, 0, \ldots, 0) = 1$ must be identical if their first order moments coincide. So, the largest of Chow parameters $\mu_{0,0,\ldots,0}(f^{-1}(1))$ can be replaced with a number from $\{0, 1\}$. This number, let us say, is chosen to be 1 when $f(0, 0, \ldots, 0) = 1$, and 0 otherwise.
- Perhaps a more important consequence of Theorem 55.1 is an analogous extension of the previous item to partially defined linear threshold functions which are also of interest [19].

Observation 55.2

The encoding linear threshold functions by Chow parameters or by their slight modification (see the previous items) that comes from our encoding scheme requires $\mathcal{O}\left(d^2\right)$ *bits per coded function and that is, because of Eq. (55.9), an optimal bit rate.*

In the rest of this section we consider a more general problem. It is known [20] that any Boolean function $\{0, 1\}^d \to \{0, 1\}$ can be represented as a threshold function of the form

$$f(x_1, x_2, \ldots, x_d) = sgn\left(\sum_{(p_1, p_2, \ldots, p_d) \in \mathcal{S} \subset \{0,1\}^d} a_{p_1, p_2, \ldots, p_d} \cdot x_1^{p_1} \cdot x_2^{p_2} \cdot \cdots \cdot x_d^{p_d}\right) \tag{55.10}$$

if $\mathcal{S}$ is allowed to be as large as $\{0, 1\}^d$. Thus, there is a big gap between the number of all threshold functions defined on $\{0, 1\}^d$, which is 2^{2^d}, and the number of linear threshold functions which is $2^{d^2 + \mathcal{O}(d^2/\log d)}$ (see Eq. (55.8) and Eq. (55.9)). Noticing that linear threshold functions are realizable by the set $\{1, x_1, x_2, \ldots, x_d\}$ of $d+1$ monomials, while the set $\{x_1^{p_1} \cdot x_2^{p_2} \cdot \ldots \cdot x_d^{p_d} \mid (p_1, p_2, \ldots, p_d) \in \{0,1\}^d\}$ consisting of 2^d monomials is required for the realization of all threshold functions, the natural question to ask is: *What happens when the number of monomials (terms) in Eq. (55.10) is somewhere between d and* 2^d*?* Let us look at the answers to these problems which follow from Theorem 55.1. We start with a formal definition.

Definition 55.3

Let a set $\mathcal{S} \subset \{0,1\}^d$ *be fixed. Then a function having the form as in Eq. (55.10) is called an* $\mathcal{S}$*-threshold function. Note that* $(0, 0, \ldots, 0) \in \mathcal{S}$ *is assumed in accordance with Eq. (55.7).*

A characterization of $\mathcal{S}$-threshold functions by using spectral coefficients has been described in Ref. [20]. There has been shown that the number of $\mathcal{S}$-functions is upper bounded by

$$2^{(d+1)\cdot \#\mathcal{S}} \qquad (55.11)$$

Here we use discrete moments to characterize $\mathcal{S}$-threshold functions. It is worth mentioning that discrete moments are easier (faster) to compute. The results are derived without restrictions to the coefficients $a_{p_1, p_2, \ldots, p_d}$ which appear in Eq. (55.10) and without restrictions on $\mathcal{S}$, either. The following theorem holds.

Theorem 55.7

Let $\mathcal{S} \subset \{0,1\}^d$ *be fixed.*

(a) *Let* $f(x_1, x_2, \ldots, x_d)$ *be an* $\mathcal{S}$*-threshold function and let* $g(x_1, x_2, \ldots, x_d)$ *be an arbitrary (not necessarily* $\mathcal{S}$*-threshold) function. Then* $f(x_1, x_2, \ldots, x_d) = g(x_1, x_2, \ldots, x_d)$ *if and only if*

$$\mu_{p_1, p_2, \ldots, p_d}(f^{-1}(1)) = \mu_{p_1, p_2, \ldots, p_d}(g^{-1}(1)) \qquad \textit{for all} \quad (p_1, p_2, \ldots, p_d) \in \mathcal{S}$$

(b) *Let* $f(x_1, x_2, \ldots, x_d)$ *and* $g(x_1, x_2, \ldots, x_d)$ *be* $\mathcal{S}$*-threshold functions. Then* $f(x_1, x_2, \ldots, x_d) = g(x_1, x_2, \ldots, x_d)$ *if and only if* $f(0, 0, \ldots, 0) = g(0, 0, \ldots, 0)$ *and*

$$\mu_{p_1, p_2, \ldots, p_d}(f^{-1}(1)) = \mu_{p_1, p_2, \ldots, p_d}(g^{-1}(1)) \qquad \textit{for all} \quad (p_1, p_2, \ldots, p_d) \in \mathcal{S} \setminus (0, 0, \ldots, 0)$$

(c) *There are at most*

$$4 \cdot 2^{d\cdot(\#\mathcal{S}-1) - \sum_{(p_1, p_2, \ldots, p_d) \in \mathcal{S}} (p_1 + p_2 + \cdots + p_d)} \qquad (55.12)$$

different $\mathcal{S}$*-threshold functions defined on* $\{0,1\}^d$*.*

Proof

(a) It is easy to see that any $\mathcal{S}$-threshold function $f(x_1, x_2, \ldots, x_d)$ separates $f^{-1}(1) \setminus B$ and $B \setminus f^{-1}(1)$ for any $B \subset \{0,1\}^d$. Let us set $B = g^{-1}(1)$, then the statement follows from Theorem 55.1.

(b) Let $f(x_1, x_2, \ldots, x_d) = sgn\left(\sum_{(p_1, p_2, \ldots, p_d) \in \mathcal{S}} a_{p_1, p_2 \ldots, p_d} \cdot x_1^{p_1} \cdot x_2^{p_2} \cdot \cdots \cdot x_d^{p_d}\right)$ and $g(x_1, x_2, \ldots, x_d) = sgn\left(\sum_{(p_1, p_2, \ldots, p_d) \in \mathcal{S}} b_{p_1, p_2 \ldots, p_d} \cdot x_1^{p_1} \cdot x_2^{p_2} \cdot \cdots \cdot x_d^{p_d}\right)$.

The equality $f(x_1, x_2, \ldots, x_d) = g(x_1, x_2, \ldots, x_d)$ preserves $\frac{b_{0,0,\ldots,0}}{a_{0,0,\ldots,0}} > 0$, implying that the function

$$h(x_1, x_2, \ldots, x_d) = \frac{b_{0,0,\ldots,0}}{a_{0,0,\ldots,0}} \left(\sum_{(p_1, p_2, \ldots, p_d) \in \mathcal{S}} a_{p_1, p_2 \ldots, p_d} \cdot x_1^{p_1} \cdot x_2^{p_2} \cdot \ldots \cdot x_d^{p_d} \right)$$

$$- \left(\sum_{(p_1, p_2, \ldots, p_d) \in \mathcal{S}} b_{p_1, p_2 \ldots, p_d} \cdot x_1^{p_1} \cdot x_2^{p_2} \cdot \ldots \cdot x_d^{p_d} \right)$$

separates $f^{-1}(1) \setminus g^{-1}(1)$ and $g^{-1}(1) \setminus f^{-1}(1)$—it is easy to prove the implications

$(\overline{x}_1, \ldots, \overline{x}_d) \in f^{-1} \setminus g^{-1}(1) \Rightarrow h(\overline{x}_1, \ldots, \overline{x}_d) > 0$ and

$(\overline{x}_1, \ldots, \overline{x}_d) \in g^{-1} \setminus f^{-1}(1) \Rightarrow h(\overline{x}_1, \ldots, \overline{x}_d) < 0$

By noticing that the separating function $h(x_1, x_2, \ldots, x_d)$ is of the form (55.1) and specifying $\mathcal{P} = \mathcal{S}$ it follows from Theorem 55.1 that $f^{-1}(1)$ and $g^{-1}(1)$ must be equal, or equivalently, $f(x_1, \ldots, x_d) = g(x_1, \ldots, x_d)$ for all $(x_1, \ldots, x_d) \in \{0, 1\}^d$.

(c) The proof follows from the bijection established in (b). An upper bound for the number of different $(\#\mathcal{S} - 1)$-tuples made by $\mu_{p_1, p_2, \ldots, p_d}(f^{-1}(1))$ (with $(p_1, p_2, \ldots, p_d) \in \mathcal{S} \setminus (0, 0, \ldots, 0)$ and $f(0, 0, \ldots, 0) = 0$) follows from the implication:

$$p_1 + p_2 + \cdots + p_d = i \;\Rightarrow\; \mu_{p_1, p_2, \ldots, p_d}(f^{-1}(1)) \in \begin{cases} [0, 2^{d-i}) & \text{if} \quad f(1, 1, \ldots, 1) = 0 \\ (0, 2^{d-i}] & \text{if} \quad f(1, 1, \ldots, 1) = 1 \end{cases}$$

for all $(p_1, p_2, \ldots, p_d) \in \{0, 1\}^d$ and $i = 0, 1, \ldots, d$. Therefore,

- the number of $\mathcal{S}$-threshold functions which map $(1, 1, \ldots, 1) \to 1$ and $(0, 0, \ldots, 0) \to 0$ and
- the number of $\mathcal{S}$-threshold functions which map $(1, 1, \ldots, 1) \to 0$ and $(0, 0, \ldots, 0) \to 0$ are both upper bounded with

$$\prod_{(p_1, p_2, \ldots, p_d) \in \mathcal{S} \setminus (0,0,\ldots,0)} 2^{d-(p_1+p_2+\cdots+p_d)} = 2^{d\cdot(\#\mathcal{S}-1) - \sum_{(p_1, p_2, \ldots, p_d) \in \mathcal{S}} (p_1+p_2+\cdots+p_d)}$$

which proves (c). □

Note that the proved bound for the number of $\mathcal{S}$-threshold functions improves (55.11).

55.5 Concluding Remarks

The problem of partitioning a finite d-dimensional squared (integer) grid appears in many areas of computer science and mathematics. This chapter considered just some of them. We demonstrated how a subset of such an integer grid can be represented uniquely by a suitably chosen set of discrete moments. In the case of planar digital objects, we characterized situations when the least squares fit polynomials preserve a unique identification of the fitted object from the considered set of digital objects. This is a particular advantage in the area of image processing and digital image analysis. The same method can be applied in three dimensions as well. Even though the 3D case was discussed briefly, let us mention that the characterization by discrete moments described here is important because there are not many ways to characterize discrete point sets in 3D. Indeed, two sets could be compared by looking at their convex hulls but this could be inappropriate because very simple objects, such as digital tetrahedrons, for example, could have a large number of vertices on their convex hull—for details see Ref. [21]. Of course, the Freeman encoding has not a straightforward extension in 3D. The application of our method to multilevel threshold functions defined on not necessarily binary inputs (see Ref. [22]) was not analyzed in this chapter, but we will analyze it in the near future. To make the discussion more complete, let us give an example when the method does not apply well. If we consider the set of digital convex polygons on an $m \times m$ integer grid, then Theorem 55.1 suggests a large number of moments that have to be used. That is far from the optimum. The result of Ref. [23] suggests that the encoding of such digital convex polygons can be done within an $\mathcal{O}(m^{2/3})$ bit rate per encoded polygon. Let us point out that some versions of the problem (the infinite grid $\mathbb{N}_0^d$ should be divided) can be found in some areas where they are perhaps not expected—in the group theory [24–26], for example.

Our results may be viewed as a technique for representing planar digital objects and threshold functions in near-optimal space. In this sense it is a heuristic for minimizing the space required to represent these objects.

References

[1] Koplowitz, J., Lindenbaum, M., and Bruckstein, A., The number of digital straight lines on $n \times n$ grid, *IEEE Trans. Inf. Theor.*, 36, 192, 1990.

[2] Lindenbaum, M. and Koplowitz, J., A new parametrization of digital straight lines, *IEEE Trans. Pattern Anal. Mach. Intell.*, 13, 847, 1991.

[3] Žunić, J. and Acketa, D. M., A general coding scheme for families of digital curve segments, *Graphical Models Image Process.,* 60, 437, 1998.

[4] Freeman, H., Boundary encoding and Processing, in *Picture Processing and Psyhopictorics,* Lipkin, B. S. and Rosenfeld, A., Eds., Academic Press, New York, 1970, p. 391.

[5] Melter, R. A. and Rosenfeld, A., New views of linearity and connectedness in digital geometry, *Pattern Recognition Lett.*, 10, 9, 1989.

[6] Melter, R. A., Stojmenović, I., and Žunić, J., A new characterization of digital lines by least square fits, *Pattern Recognition Lett.*, 14, 83, 1993.

[7] Klette, R. and Žunić, J., Multigrid convergence of calculated features in image analysis, *J. Math. Imaging Vision,* 13, 173, 2000.

[8] Klette,R., Stojmenović, I., and Žunić, J., A new parametrization of digital planes by least squares plane fit and generalizations, *Graphical Models Image Process.,* 58, 295, 1996.

[9] Žunić, J. and Sladoje, N., Efficiency of characterizing ellipses and ellipsoids by discrete moments, *IEEE Trans. Pattern Anal. Machine Intell.,* 22, 407, 2000.

[10] Chow, C. K., Boolean functions realizable with single threshold devices, *Proc. IRE*, 49, 370, 1960.

[11] Chow, C. K., On the characterization of threshold functions, in *Switching Circuit Theory and Logical Design*, IEEE Special Publication S-134, 1961, p. 34.

[12] Siu, K.-Y., Roychowdhury, V., and Kailtah, T., *Discrete Neural Computation A Theoretical Foundation,* Prentice-Hall, New Jersey, 1995.

[13] Muroga, S., Lower bounds on the number of threshold functions and a maximum weight, *IEEE Trans. Electron. Comput.*, EC-14, 136, 1965.

[14] 'Olafsson, S. and Abu-Mostafa, Y. S., The capacity of multilevel threshold functions, *IEEE Trans. Pattern Anal. Machine Intell.,* 10, 277, 1988.

[15] Hopfield, J. J., Neural networks and physical systems with emergent collective computational abilities, *Proc. Nat. Acad. Sci. USA*, 79, 2554, 1982.

[16] Winder, R. O., Chow parameters in threshold logic, *JACM,* 18, 265, 1971.

[17] Zuev, Y. A., Asymptotics of the logarithm of the number of threshold functions of the algebra of logic, *Sov. Math. Dok.,* 39, 512, 1989.

[18] Žunić, J., On encoding and enumerating threshold functions, *IEEE Trans. Neural Networks,* 15, 261, 2004.

[19] Winder, R. O., Threshold logic asymptotes, *IEEE Trans. Comput.,* C-19, 349, 1970.

[20] Bruck, J., Harmonic analysis of polynomial threshold functions, *SIAM J. Disc. Math.,* 3, 168 1990.

[21] Hayes, A. S. and Larman, D. C., The vertices of the knapsack polytope, *Disc. Appl. Math.,* 6, 135, 1983.

[22] Obradović, Z. and Parberry, I., Computing with discrete multivalued neurons, *JCSS*, 45, 471, 1992.

[23] Ivić, A., Koplowitz, J., and Žunić, J., On the number of digital convex polygons inscribed into an (m,m)-grid, *IEEE Trans. Inf. Theor.* 40, 1681, 1994.

[24] Corteel, S., Rémond, G., Schaeffer, G., and Thomas, H., The number of plane corner cuts, *Adv. Appl. Math.*, 23, 49, 1999.

[25] Onn, S. and Sturmfels, B., Cutting corners, *Adv. Appl. Math.*, 23, 29, 1999.

[26] Wagner, U., On the number of corner cuts, *Adv. Appl. Math.*, 29, 152, 2002.

56
Maximum Planar Subgraph

Gruia Calinescu*
Illinois Institute of Technology

Cristina G. Fernandes†
University of São Paulo

56.1 Introduction

In graph theory, questions related to planarity always played an important role. The main subject of this chapter is the following problem, which is denoted here as MAXIMUM-WEIGHT PLANAR SUBGRAPH: Given a graph G with a nonnegative weight defined for each edge, find a planar subgraph of G of maximum weight, where the weight of a subgraph is simply the sum of the weights of the edges in the subgraph. Its unweighted version, denoted as MAXIMUM PLANAR SUBGRAPH, is: given a simple graph G, find a planar subgraph of G with the maximum number of edges.

These problems have applications in circuit layout [1–6], facility layout [5,7], and graph drawing [8–10](the process of drawing a graph, usually on a two-dimensional medium, as "nicely" as possible, where "nicely" is defined by the application). For example, graph drawing of nonplanar graphs can start by drawing a planar subgraph [11,12], and then drawing the remaining edges. This method employs planarization, the process of obtaining a large planar subgraph from a nonplanar one [13–15].

Unfortunately, MAXIMUM PLANAR SUBGRAPH is NP-hard [16,17]. In fact, MAXIMUM PLANAR SUBGRAPH is known to be Max SNP-hard [18], which means there is an $\epsilon > 0$ such that no approximation algorithm achieves a ratio of $1 - \epsilon$ for it, unless P = NP. This is true even if the input is a cubic graph [19]. The largest ϵ for which this result is known however is tiny, making $1 - \epsilon$ far from the best approximation ratios known for the problem.

On the positive side, for years, the best known approximation algorithm for MAXIMUM PLANAR SUBGRAPH was the trivial one that produces a spanning tree of the (connected) input graph. Using Euler's formula, it is easy to see that this algorithm achieves a ratio of 1/3. There were several heuristics proposed for MAXIMUM PLANAR SUBGRAPH and MAXIMUM-WEIGHT PLANAR SUBGRAPH, but most of them either have a ratio of 1/3 or were not approximation algorithms at all [20]. In particular, the natural MST algorithm for MAXIMUM-WEIGHT PLANAR SUBGRAPH, which outputs a maximum weight forest of the given graph, has also a ratio of 1/3. Only in the 1990s, the 1/3 threshold was broken, by algorithms that use *triangular structure* (a graph whose all blocks are edges or triangles) [18,21]. Since then, to our knowledge, no better approximation algorithms appeared in the literature for these problems.

*G. Calinescu's work was supported by NSF Grant CCF-0515088.

†C. G. Fernandes' work was supported by ProNEx - FAPESP/CNPq Proc. No. 2003/09925-5.

Specifically, the best known approximation algorithm for MAXIMUM PLANAR SUBGRAPH is the one that outputs a triangular structure of the input graph with the maximum number of edges. Such a triangular structure can be computed in polynomial time by a sophisticated algorithm for polymatroid matching. The approximation ratio of the resulting algorithm is 4/9 [18]. Its analysis and implementation however are intricate. A simpler and faster greedy algorithm that also produces a triangular structure achieves a ratio of 7/18 [18].

As for MAXIMUM-WEIGHT PLANAR SUBGRAPH, the best approximation ratio known is $1/3 + 1/72$ [21]. Triangular structures are related to 3-restricted Steiner trees, which are used to achieve good approximation ratios for the famous MINIMUM STEINER TREE problem (see Chapter 42). The ratio of $1/3 + 1/72$ for MAXIMUM-WEIGHT PLANAR SUBGRAPH is achieved by an algorithm that follows closely an algorithm by Berman and Ramaiyer [22] for MINIMUM STEINER TREE. The output is also a triangular structure. A randomized pseudopolynomial algorithm that computes a maximum-weight triangular structure in a given weighted graph can be derived from a result of Camerini et al. [23]. This algorithm can be converted, using the standard method of rounding and scaling (as in Chapters 9–11, or in Ref. [24, pp. 135–137]), to a $(1 - \varepsilon)$-approximation algorithm for computing a maximum-weight triangular structure. A $(1 - \varepsilon)$-approximation parallel algorithm with polylogarithmic running time also follows from a result by Prömel and Steger [25]. A maximum-weight triangular structure also achieves a $1/3 + 1/72$ ratio, however we know of no direct proof for this. The only proof we know is based on the algorithm that mimics Berman and Ramaiyer's.

In this chapter, we describe the main approximation algorithms known for both MAXIMUM PLANAR SUBGRAPH and MAXIMUM-WEIGHT PLANAR SUBGRAPH. We also discuss the relation between these problems and MINIMUM STEINER TREE.

The chapter is organized as follows. Section 56.2 analyzes the Maximum Spanning Tree (MST) algorithm. Section 56.3 is about triangular structures, their relation to 3-restricted Steiner trees and the above-mentioned algorithms that generate triangular structures. Section 56.4 presents the analysis of some of these triangular structure algorithms. Section 56.5 shows the results of applying the same methods to the MAXIMUM OUTERPLANAR SUBGRAPH problem, where the output graph must be outerplanar (i.e., can be drawn in the plane with all vertices on the boundary of the outer face). Due to the importance of these problems, exact algorithms with exponential worst-case running time have been proposed, and heuristics have been analyzed experimentally. We discuss these practical approaches in Section 56.6 together with related problems and results and ideas for improved approximation ratios.

56.2 Analysis of the MST Algorithm

The MST algorithm, given a graph G and a nonnegative weight for each of its edges, outputs a maximum weight forest of G. We know two ways to demonstrate that this algorithm has a ratio of 1/3 for MAXIMUM-WEIGHT PLANAR SUBGRAPH. One of them uses an idea of Korte and Hausmann [26]. The other one uses a well-known theorem of Nash-Williams involving the following concept.

The *arboricity* of a graph is the minimum number of spanning forests into which its edge set can be partitioned. Nash-Williams [27,28] proved the following classic theorem about it.

Theorem 56.1

The arboricity of a graph G is the maximum, over all subgraphs H of G with at least two vertices, of

$$\left\lceil \frac{|E(H)|}{|V(H)| - 1} \right\rceil$$

For a function w defined on the edge set of a graph G and a set S of edges of G, we use $w(S)$ to denote the sum of the weights of the edges in S and, for a subgraph H of G, we use $w(H)$ instead of $w(E(H))$. A family of graphs is said to be *closed under taking subgraphs* if and only if any subgraph of any graph in the family is in the family. The next lemma is the key in the analysis of the approximation ratio of the MST algorithm.

Lemma 56.1

Let $\mathcal{F}$ be a family of graphs closed under taking subgraphs, such that, for some positive integer $c, |E(G)| \leq c(|V(G)| - 1)$ for all G in $\mathcal{F}$. Let w be a nonnegative weight function on the edges of some G in $\mathcal{F}$, and F be a maximum-weight forest in G. Then $w(F) \geq \frac{1}{c} w(G)$.

First Proof

Let G be a graph in $\mathcal{F}$ and w be a nonnegative weight function on its edges. We have $|E(H)| \leq c(|V(H)|-1)$ for any subgraph H of G, because $\mathcal{F}$ is closed under taking subgraphs. Thus, by Theorem 56.1, the arboricity of G is at most c. This means that the edge set of G can be partitioned into c forests $F_1, \ldots, F_c$. Clearly, we have $w(F_1) + \cdots + w(F_c) = w(G)$. But $w(F) \geq w(F_i)$ for all i, and this implies that $w(G) = w(F_1) + \cdots + w(F_c) \leq c \cdot w(F)$. ■

Second Proof

Let G be a graph in $\mathcal{F}$ with a nonnegative weight for each of its edges. Use Kruskal's algorithm to construct a maximum-weight forest F. That is, let $e_1, \ldots, e_m$ be the edges of G in nonincreasing order of weight. Start with $F_0 := \emptyset$. For $i = 1, \ldots, m$, if the addition of e_i to F_{i-1} creates a cycle, let $F_i := F_{i-1}$, otherwise let $F_i := F_{i-1} \cup \{e_i\}$. At the end, let $F := F_m$.

For each i, let $E_i := \{e_1, \ldots, e_i\}$ and let w_i be the weight of e_i. Let $w_{m+1} := 0$. By rearranging the terms,

$$w(F) = \sum_{i=1}^{m} |F_i|(w_i - w_{i+1}), \text{ and } w(G) = \sum_{i=1}^{m} |E_i|(w_i - w_{i+1})$$

It is therefore enough to show that $|E_i| \leq c|F_i|$ for $i = 1, \ldots, m$.

For each i, let $p_1, \ldots, p_k$ be the number of vertices in the components of F_i. Of course, $|F_i| = \sum_{j=1}^{k}(p_j - 1)$. Any edge of E_i must have its two endpoints in the same component of F_i. (Otherwise, the edge could have been selected by the algorithm, merging two components of F_i.) We associate each edge of E_i with the component of F_i that contains both of its endpoints.

The edges of E_i associated with a component of F_i are a subset of the edges of the subgraph of G induced by the vertices of this component. Thus, the number of edges associated with the jth component of F_i is at most $c(p_j - 1)$, because the subgraph of G induced by F_i is in $\mathcal{F}$, as $\mathcal{F}$ is closed under taking subgraphs. But then $|E_i| \leq \sum_{j=1}^{k} c(p_j - 1) = c|F_i|$. ■

The approximation ratio of 1/3 for the MST algorithm for MAXIMUM PLANAR SUBGRAPH follows from the next corollary.

Corollary 56.1 (See [Ref 1, Theorem 3.3])

Let P be a simple planar graph with a nonnegative weight function w defined on its edges. If F is a maximum-weight forest in P, then $w(F) \geq \frac{1}{3} w(P)$.

Proof

By Euler's formula, $|E(P)| \leq 3(|V(P)| - 1)$. Then the corollary follows from applying Lemma 56.1 to the family of simple planar graphs, with $c = 3$. ■

Indeed, if F is a maximum-weight forest in a graph G with a weight function w on its edges and P is any planar subgraph of G, then $w(F) \geq \frac{1}{3} w(P)$ by Corollary 56.1. This implies that the MST algorithm has a ratio of at least 1/3 for MAXIMUM-WEIGHT PLANAR SUBGRAPH. The 1/3 bound is tight, as is the case when the input is an unweighted triangulated planar graph.

If one uses a Kruskal-like heuristic where a test for planarity is performed before adding an edge, the output contains a maximum-weight forest and therefore this algorithm also has an approximation ratio of at least 1/3. Tight examples are known [1,20].

A graph is said to be *outerplanar* if it can be drawn in the plane with all vertices on the boundary of the outer face. Consider the MAXIMUM-WEIGHT OUTERPLANAR SUBGRAPH problem: Given an edge-weighted graph G, find an outerplanar subgraph of G of maximum weight. This problem is known to be NP-hard [24, p. 197]. The MST algorithm has a ratio of 1/2 for MAXIMUM-WEIGHT OUTERPLANAR SUBGRAPH, as we show below using the same techniques.

Corollary 56.2

Let P be an outerplanar graph with a nonnegative weight function w defined on its edges. Then a maximum-weight forest F of P satisfies $w(F) \geq \frac{1}{2}w(P)$.

Proof

Any subgraph of an outerplanar graph is outerplanar, and, by Euler's formula, $|E(P)| \leq 2(|V(P)| - 1)$ for any outerplanar graph P [29, Corollary 11.9]. Thus, it is enough to apply Lemma 56.1 with $c = 2$ to the family of all outerplanar graphs. ■

Let P be any outerplanar subgraph of the graph G with a weight function w on its edges and F be a maximum-weight forest. By Corollary 56.2, we deduce that $w(F) \geq \frac{1}{2}w(P)$, and the approximation ratio of 1/2 for MAXIMUM-WEIGHT OUTERPLANAR SUBGRAPH follows.

56.3 Triangular Structures and Better Approximations

We start by defining some terms and then presenting two algorithms for MAXIMUM PLANAR SUBGRAPH: algorithm A (the greedy one) and algorithm B (the currently best one).

A *triangular cactus* is a graph whose cycles (if any) are triangles and such that all edges appear in some cycle (see Figure 56.1). A *triangular cactus in a graph G* is a subgraph of G that is a triangular cactus.

A *triangular structure* is a graph whose cycles (if any) are triangles (see Figure 56.2). A *triangular structure in a graph G* is a subgraph of G that is a triangular structure. Note that every triangular cactus is a triangular structure, but not vice versa. Observe also that every triangular structure is planar, as it does not contain cycles of length greater than three.

Given a graph $G = (V, E)$ and $E' \subseteq E$, we denote by $G[E']$ the *spanning* subgraph of G induced by E', that is, the graph (V, E') (Note that this is not the usual definition of subgraph induced by an edge set, since we require the subgraph to be spanning.).

The approximation ratio of algorithm A for MAXIMUM PLANAR SUBGRAPH, as we will see, is $7/18 = 0.3888\ldots$. Algorithm A generates a triangular structure in the given graph G. As seen below, it consists of two phases. First, it greedily constructs a maximal triangular cactus S_1 in G. Second, it extends S_1 to a triangular structure S_2 in G by adding as many edges as possible to S_1 without forming any new cycles.

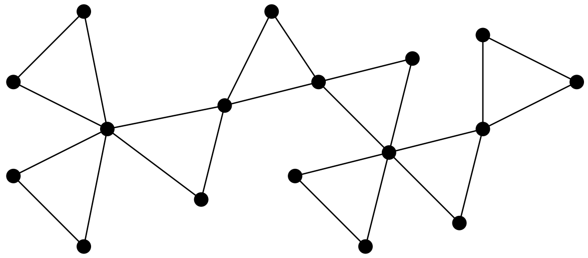


FIGURE 56.1 A (connected) triangular cactus.

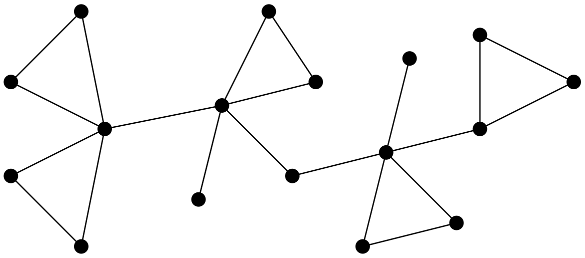


FIGURE 56.2 A (connected) triangular structure.

Algorithm **A** (G)

```
1  $E_1 \leftarrow \emptyset$
2  while there is a triangle $\Delta$ in $G$ whose vertices are in different components of $G[E_1]$ do
3       $E_1 \leftarrow E_1 \cup E(\Delta)$
4  $E_2 \leftarrow E_1$
5  while there is an edge $e$ in $G$ whose endpoints are in different components of $G[E_2]$ do
6       $E_2 \leftarrow E_2 \cup \{e\}$
7  $S_2 \leftarrow G[E_2]$
8  return $S_2$
```

Note that S_2 is indeed a triangular structure in G and therefore is a planar subgraph of G. It is easy to see that algorithm A is polynomial. Also, it can be implemented to run in linear time for graphs with bounded degree [18]. Its approximation ratio will be analyzed in the next section. Now we present the second algorithm for MAXIMUM PLANAR SUBGRAPH.

Algorithm B is very similar to algorithm A. The only difference is that, in the first phase, it finds a triangular cactus with the maximum number of triangles. This can be done in polynomial time using an algorithm for the so-called GRAPHIC MATROID PARITY problem [30,31] as a subprocedure. In the pseudocode below, MAXTRICACTUS denotes a routine that uses such an algorithm to obtain a triangular cactus in G with maximum number of triangles. See Ref. [18] for the details on how this routine does that.

Algorithm **B** (G)

```
1  $E_1 \leftarrow$ MAXTRICACTUS($G$)
2  $E_2 \leftarrow E_1$
3  while there is an edge $e$ in $G$ whose endpoints are in different components of $G[E_2]$ do
4       $E_2 \leftarrow E_2 \cup \{e\}$
5  $S_2 \leftarrow G[E_2]$
6  return $S_2$
```

Again it is clear that S_2 is a triangular structure in G and therefore is a planar subgraph of G. Gabow and Stallmann [30] described an algorithm for GRAPHIC MATROID PARITY that runs in time $O(m'n' \log^6 n')$, where m' and n' are the number of edges and vertices, respectively, in the input graph for this problem. Using this algorithm, one can get an implementation for MAXTRICACTUS that runs in time $O(m^{3/2} n \log^6 n)$, where m is the number of edges in the input graph and n is the number of vertices. So line 1 can be implemented to run in polynomial time, and again the whole algorithm has a polynomial-time implementation.

As for the weighted case, there is no polynomial-time algorithm known for GRAPHIC MATROID PARITY. However, Camerini et al. [23] proposed a randomized pseudopolynomial algorithm to solve the MATROID

PARITY problem. Using the standard method of rounding and scaling (as in Chapters 9 and 10, or in Ref. [24, pp. 135–137]), one can use their algorithm to obtain, in a weighted graph, for every $\varepsilon > 0$, a triangular structure whose weight is at least $1 - \varepsilon$ times the weight of a maximum-weight triangular structure. The running time of this procedure is polynomial in the size of the input and in $1/\varepsilon$.

Curiously, we do not know a direct way to prove that a maximum-weight triangular structure achieves a ratio greater than 1/3 for MAXIMUM-WEIGHT PLANAR SUBGRAPH. The only way we know how to prove that is by analyzing another algorithm—a greedy one, inspired by the algorithm of Berman and Ramaiyer [22] for MINIMUM STEINER TREE. This algorithm takes advantage of an interesting relation between triangular structures and a particular type of Steiner trees. To explain a bit of this relation, we need to introduce some notation for Steiner trees.

Let G be a connected graph and R be a set of vertices of G, usually called *terminals*. Each vertex not in R is called a *Steiner vertex*. A *Steiner tree* is a tree in G containing all terminals. We may assume that all leaves in a Steiner tree are terminals. In the MINIMUM STEINER TREE problem, we are given a graph G, a nonnegative weight for each edge of G and a set R of vertices of G, and we want to find a minimum-weight Steiner tree in G.

A *full component* of a Steiner tree T is a maximal subtree of T whose internal vertices are all Steiner vertices. For an integer k, a Steiner tree is *k-restricted* if all of its full components have at most k leaves.

Berman and Ramaiyer [22] proposed an approximation algorithm for MINIMUM STEINER TREE that generates a 3-restricted Steiner tree of weight at most 11/6 times the minimum weight of a Steiner tree. Their algorithm works for larger values of k, producing k-restricted Steiner trees having better approximation ratio. The 3-restricted version has been applied to MAXIMUM-WEIGHT PLANAR SUBGRAPH, and we give the intuition behind this approach below.

There is a close relation between triangular structures in a graph G and 3-restricted Steiner trees in an auxiliary graph H defined as follows. The set of vertices of H is $V(G)$ plus a new vertex for each triangle in G. The edge set of H is $E(G)$ plus three new edges incident to each triangle vertex. The edges incident to a triangle vertex have as the other endpoints the three vertices of the triangle in G. This completes the description of graph H. Let $V(G)$ be the set of terminals R.

Now, suppose we are given a nonnegative weight for each edge in G and let M be 10 times the largest such weight. In H, an edge of $e \in E(G)$ has weight $w'(e) = M - w(e)$. For a triangle xyz of G, if u is the new vertex of H corresponding to xyz, then we set $w'(xu) = w'(yu) = w'(zu) = 2M/3 - (w(xy) + w(xz) + w(yz))/3$. It is easy to check that a triangular structure of weight W in G corresponds to a 3-restricted Steiner tree in H of weight $(n-1)M - W$, where $n = |V(G)|$. Thus a maximum-weight triangular structure in G corresponds to a minimum-weight 3-restricted Steiner tree in H.

The reduction above does not preserve approximation ratios. Nevertheless, some algorithms designed for MINIMUM STEINER TREE can be adapted to MAXIMUM-WEIGHT PLANAR SUBGRAPH. In particular, the Berman and Ramaiyer algorithm is used by Călinescu et al. [21] to break the 1/3 threshold for MAXIMUM-WEIGHT PLANAR SUBGRAPH.

56.4 Analysis of the Triangular Structure Algorithms

We start by settling the approximation ratio of algorithm A.

Theorem 56.2

Algorithm A has approximation ratio of 7/18 *for* MAXIMUM PLANAR SUBGRAPH.

Proof

First let us show that the approximation ratio is at least 7/18. Without loss of generality, we may assume G is connected, and has at least three vertices. Observe that the number of edges in S_2 is the number of edges in a spanning tree of G plus the number of triangles in S_1. So it suffices to count the number of triangles in S_1.

Let H be a plane embedding of a maximum planar subgraph of G. Let $n \geq 3$ be the number of vertices in G, and $t \geq 0$ be such that $3n - 6 - t$ is the number of edges in H. We can think of t as the number of edges missing for H to be a triangulated plane graph. The number of triangular faces in H is at least $2n - 4 - 2t$. (This is a lower bound on the number of triangular faces of H since if H were triangulated, it would have $2n - 4$ triangular faces, and each missing edge can destroy at most two of these triangular faces.)

Let k be the number of components of S_1 each with at least one triangle, and let $p_1, p_2, \ldots, p_k$ be the number of triangles in each of these components. Let $p = \sum_{i=1}^{k} p_i$. We will prove that p, the number of triangles in S_1, is at least a constant fraction of $n - 2 - t$. Note that if a triangle cannot be added to S_1, it is because two of its vertices are in the same component of S_1. Hence one of its edges has its two endpoints in the same component of S_1. This means that, at the end of the first phase, every triangle in G must have some two vertices in the same component of S_1. In particular, every triangular face in H must have some two vertices in the same component of S_1, and therefore one of its three edges must be in the subgraph of H induced by the vertices in a component of S_1. Thus we can associate with each triangular face F in H an edge e in F whose endpoints are in the same component of S_1. But any edge e in H lies in at most two triangular faces of H, so e could have been chosen by at most two triangular faces of H. It follows that the number of triangular faces in H is at most twice the number of edges in H whose endpoints are in the same component of S_1.

Let H' be the subgraph of H induced by the edges of H whose endpoints are in the same component of S_1. Note that $p_i \geq 1$, for all i, and that the number of vertices in the ith component of S_1 is $2p_i + 1 \geq 3$. Since H' is planar, H' has at most $\sum_{i=1}^{k}(3(2p_i + 1) - 6) = 6p - 3k$ edges. By the observation at the end of the previous paragraph, $2(6p - 3k) \geq 2|E(H')| \geq$ (number of triangular faces in H) $\geq 2n - 4 - 2t$. From this, we have

$$p \geq \frac{n - 2 - t + 3k}{6} \geq \frac{n - 2 - t}{6}$$

Therefore, the number of triangles in S_1 is at least $\frac{n-2-t}{6}$, and the ratio between the number of edges in S_2 and the number of edges in H is at least

$$\frac{n - 1 + \frac{n-2-t}{6}}{3n - 6 - t} = \frac{7n - 8 - t}{18n - 36 - 6t} \geq \frac{7}{18}$$

since $t \geq 0$. This completes the proof that the approximation ratio of algorithm A is at least 7/18.

Now, we will prove that the approximation ratio is at most 7/18. Let S be any connected triangular cactus with $p > 0$ triangles. Note that S has $2p + 1 \geq 3$ vertices. Let S' be any triangulated plane supergraph of S on the same set of vertices (S' can be obtained from S by adding edges to S until it becomes triangulated). Since S' is triangulated, S' has $2(2p + 1) - 4 = 4p - 2$ (triangular) faces. For each face of S', add a new vertex in the face and adjacent to all vertices on the boundary of that face. Let G be the new graph. Observe that G is a triangulated plane graph and has $(2p + 1) + (4p - 2) = 6p - 1$ vertices. This means that G has $3(6p - 1) - 6 = 18p - 9$ edges. With G as input for algorithm A, in the first phase it can produce $S_1 = S$, and S_2 can be S plus one edge for each of the new vertices (the vertices in G not in S). The number of edges in S is $3p$. Hence S_2 can have $3p + (4p - 2) = 7p - 2$ edges, while G has $18p - 9$ edges. Thus, the ratio between the number of edges in S_2 and the number of edges in G is

$$\frac{7p - 2}{18p - 9}$$

By choosing p as large as we wish, we get a ratio as close to 7/18 as we want. ■

Algorithm B has an approximation ratio of $4/9 = 0.444\ldots$. The proof that its ratio is at least 4/9 is a consequence of a result on triangular structures in planar graphs.

For a graph H, denote by $\text{mts}(H)$ the number of edges in a triangular structure in H with the maximum number of edges. Then we have the following.

Theorem 56.3

If H is a planar graph, then $\text{mts}(H)/|E(H)| \geq 4/9$.

This theorem however has a long and complicated proof [18] (Raz [32] also announced a proof), so we chose to present here a weaker version of the result, that has a nice proof.

Theorem 56.4

If H is a planar graph, then $\text{mts}(H)/|E(H)| \geq 0.4$.

Proof

The theorem is easily verified if H has less than three vertices, so let us assume that H has $n \geq 3$ vertices. We may furthermore assume that H is connected. Embed H in the plane without crossings. Let t be such that $|E(H)| = 3n - 6 - t$. Clearly $t \geq 0$.

Now let J be any triangular cactus obtained by choosing triangular faces of H until no more can be added; say the final J has k components. Let p be the number of triangles in J. As in the proof of Theorem 56.2, if we count twice every edge in H whose endpoints are in the same component of J, we will "cover" every triangular face of H. In fact, each triangular face of J will be covered three times, by the three edges bounding the face. Let s be the number of edges in H whose endpoints are in the same component of J. Let l be the number of triangular faces in H. Since the p triangles in J are covered three times, we have $(l - p) + 3p = l + 2p \leq 2s$. As in Theorem 56.2, we have $s \leq 6p - 3k$ and $l \geq 2n - 4 - 2t$.

It follows that $2n - 4 - 2t + 2p \leq l + 2p \leq 2s \leq 2(6p - 3k)$, so that

$$p \geq \frac{2n - 4 - 2t + 6k}{10} = \frac{n - 2 - t + 3k}{5} \geq \frac{n - 2 - t}{5}$$

As $\text{mts}(H) \geq (n - 1) + p$, we have

$$\frac{\text{mts}(H)}{|E(H)|} \geq \frac{n - 1 + \frac{n-2-t}{5}}{3n - 6 - t} = \frac{6n - 7 - t}{15n - 30 - 5t} \geq \frac{2}{5} = 0.4$$ ■

Now we proceed with the analysis of algorithm B.

Theorem 56.5

Algorithm B has approximation ratio of $4/9$ *for* MAXIMUM PLANAR SUBGRAPH.

Proof

To show that the approximation ratio of algorithm B is at least 4/9, it suffices to apply Theorem 56.3 to a maximum planar subgraph of the input graph of algorithm B. (Theorem 56.4 implies a lower bound of 0.4 on the approximation ratio.)

Now, let us prove that the approximation ratio is at most 4/9. Let G' be any triangulated plane graph on n' vertices. Call V' the vertex set of G'. Since G' is triangulated, G' has $2n' - 4$ (triangular) faces. For each face of G', add a new vertex in the face, adjacent to all three vertices on the boundary of that face. Let G be the new graph and let V be the vertex set of G.

Observe that G is a triangulated plane graph, and has $n' + (2n' - 4) = 3n' - 4$ vertices. Therefore G has $3(3n' - 4) - 6 = 9n' - 18$ edges. Let S be a maximum triangular structure in G. Any edge in G has at least one endpoint in V'. Moreover $|V'| = n'$. Therefore a maximum matching in G has at most n' edges (each with at least one distinct endpoint in V'). The following lemma is observed in Ref. [33, p. 440].

Lemma 56.2

If S is a triangular structure with t triangles in a given graph G, then there is a matching in G of size t.

Using Lemma 56.2, we conclude that S has at most n' triangles. Recall that S, being a triangular structure, is a spanning tree of G plus one edge per triangle in S, which implies that S has at most

$(3n' - 5) + n' = 4n' - 5$ edges. Furthermore, G has $9n' - 18$ edges. Therefore, the ratio between the number of edges in S and the number of edges in G is

$$\frac{4n' - 5}{9n' - 18}$$

By choosing n' as large as we wish, we get a ratio as close to 4/9 as required. ■

Surprisingly, somehow Theorem 56.3 does not extend to the weighted case. Let mwts(G) denote the weight of a maximum-weight triangular structure in a weighted graph G. Then we have the following.

Theorem 56.6

If H is a planar graph, then $\text{mwts}(H)/w(H) \geq 1/3 + 1/72$.

This theorem, however, has a long and complicated proof [21] based on the algorithm adapted from the Berman and Ramaiyer MINIMUM STEINER TREE algorithm. The bound is not tight, and in fact we conjecture the following.

Conjecture 56.1

If H is a planar graph, then $\text{mwts}(H)/w(H) \geq 5/12$.

The next theorem shows that Conjecture 56.1 is as strong as possible in the sense that it does not hold with a constant bigger than 5/12. Conjecture 56.1, if true, would imply that the algorithm that produces a maximum-weight triangular structure in the input weighted graph has an approximation ratio of 5/12.

Theorem 56.7

For any $\epsilon > 0$, there is a planar graph H and a weight function w defined on its edges for which $\text{mwts}(H)/w(H) \leq 5/12 + \epsilon$.

Proof

Let P be any triangulated plane graph with $n \geq 3$ vertices. By Euler's formula, P has $3n - 6$ edges. Let the weight of each edge of P be two. In each of the $2n - 4$ faces of P, add a new vertex and three edges adjacent to this new vertex and the three vertices of P that define the face. These new edges have weight one. Denote by H the plane graph obtained this way and by w the weight function defined on its edges.

Observe that H has $3n - 4$ vertices, is triangulated, and has total weight $2 \cdot [3n - 6] + 1 \cdot [3(2n - 4)] = 12n - 24$ (see Figure 56.3). Let us prove that a maximum-weight triangular structure in H has weight at most $5/12 + o(1)$ of the weight of H.

Consider an arbitrary triangular structure S in H. Let S_1 be the set of edges of weight one in S and S_2 be the remaining edges of S (all of weight two).

First, we may assume S connects $V(H)$, as this can be obtained without decreasing the weight of S. Second, we may assume that S_2 connects $V(P)$. Indeed, assume this is not the case, and let $V_1, \ldots, V_k$ be the connected components of $P[S_2]$. Then there are two such components, say V_1 and V_2, such that two edges e_1 and e_2 in S_1 form a path of length two from a vertex v_1 in V_1 to a vertex v_2 in V_2. Indeed this is the only way to connect vertices of P with edges from S_1. The middle point of this path sits in a face of P so let v_3 be the third vertex of P, besides v_1 and v_2, bordering this face. By interchanging, if necessary, indexes 1 and 2 we may assume that $v_3 \notin V_2$. Then the weight-2 edges v_1v_2 and v_2v_3 do not belong to S_2. As S only contains cycles of length three, we deduce that e_2 (which is incident to v_2) does not belong to any cycle of S. Thus replacing e_2 by v_1v_2 results in a triangular structure of larger weight.

Now we divide the edges of S_2 into two sets: set X, containing edges that belong to triangles of S_2, and set Y, containing the remaining edges. Removing exactly one edge from each triangle of S_2 leaves us with a spanning tree of P, and therefore $2|X|/3 + |Y| = n - 1$. Each triangle of S that is not in X must contain one edge from Y, or we have cycles of length 4. Now recall that S_2 connects P. Thus for every vertex of H not in P, we have in S exactly one or two edges, and if we have two edges, we have a triangle of S with one

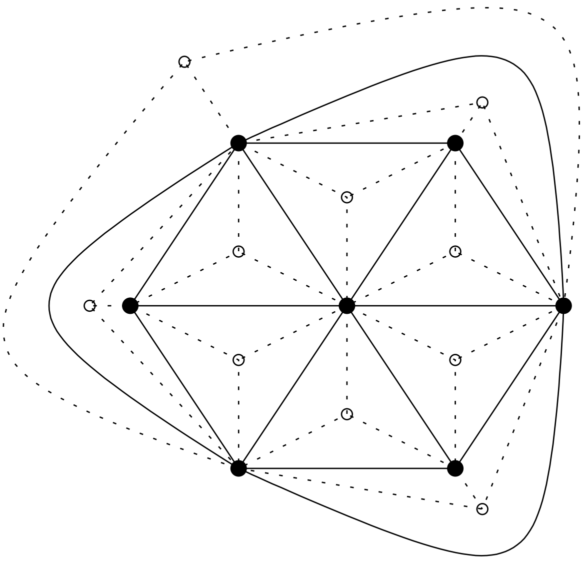


FIGURE 56.3 An example of graph H. The solid edges show graph P. Each of them has weight two. The dotted part shows the extra vertices and edges in H. The dotted edges have weight one.

edge in Y. One edge of Y can appear in at most one triangle, as otherwise we get a cycle of length four. In conclusion, we have

$$w(S) \leq (|X|+|Y|)\cdot 2+(2n-4)\cdot 1+|Y|\cdot 1 \leq 3\left(\frac{2}{3}|X|+|Y|\right)+2n-4 = 3(n-1)+2n-4 = 5n-7$$

Therefore, as n goes to ∞, the ratio $\text{mwts}(H)/w(H)$ approaches $5/12$ and the theorem follows. ■

A consequence of Theorem 56.7 is that the approximation ratio of any algorithm for MAXIMUM-WEIGHT PLANAR SUBGRAPH that produces a triangular structure in the input graph is at most 5/12. In particular, the approximation ratio of the algorithm that mimics Berman and Ramaiyer's and the one that produces a maximum-weight triangular structure in the input weighted graph is at most 5/12.

56.5 Outerplanar Subgraphs

A triangular structure is an outerplanar graph. We will show that a maximum-weight triangular structure in a weighted graph G has weight at least two thirds of the weight of a maximum-weight outerplanar subgraph of G. This implies a $(2/3 - \epsilon)$-approximation algorithm for MAXIMUM-WEIGHT OUTERPLANAR SUBGRAPH and, for the unweighted version of the problem, a 2/3-approximation algorithm (via algorithm B, described in Section 56.3).

The proof of the next key lemma takes a few pages.

Lemma 56.3

In any maximal outerplanar graph P, there are at most three (not necessarily distinct) triangular structures in P such that each edge of P appears in exactly two of them.

Proof

If P has fewer than three vertices, then P and the empty graph are triangular structures. Let us assume P has at least three vertices. Embed P in the plane as a triangulation of a polygon. Every maximal outerplanar graph with at least three vertices is a triangulation of a polygon. That is, the boundary of the exterior face

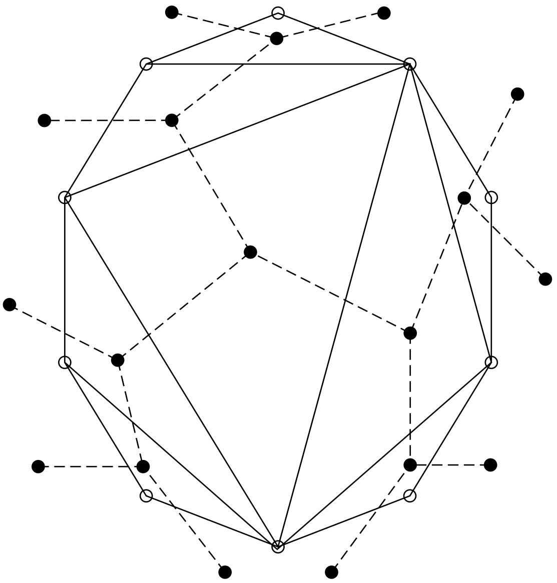


FIGURE 56.4 An outerplanar graph (solid lines) and the tree D' (filled vertices and dotted lines) obtained from its dual.

of P is a Hamiltonian cycle H and each interior face is triangular [29, p. 106]. Let E be the edge set of P, b be the exterior face of P, and F be the set of faces of P other than b.

Let D be the dual multigraph of P. Let us call the vertices of D (which are faces of P) *nodes*, and the edges of D, *arcs*. All nodes of D but b have degree three. Also, the edges in the Hamiltonian cycle H correspond to the arcs incident to b in D.

Let D' be the graph obtained from D by subdividing each arc incident to b, and then removing b. See Figure 56.4 for an example.

Lemma 56.4

D' is a tree all of whose internal nodes have degree three.

Proof

First, let us prove that D' has no cycle. It is enough to show that any cycle in D contains b. A cycle in D corresponds to a cut in P [34, p. 143, ex. 9.2.3]. Because H is a Hamiltonian cycle, any cut in P contains at least two edges of H, which correspond to arcs incident to b. Therefore, any cycle in D contains at least two arcs incident to b, so it contains b.

Second, let us prove that D' is connected. If D' were not connected, there would be two nodes u and v in different components of D'. Let us argue that we can choose u and v to be nodes in $V(D)$. If u were not a node in $V(D)$, then it would be a node that originated from the subdivision of an arc incident to b, and thus it would have degree one in D'. Change u to its unique neighbor in D'. Do the same for v. Note that the new u and v must still be in different components of D', since they are in the same component as the initial u and v, respectively. So we can assume u and v are nodes of D. And because D is connected, there is a path in D between u and v. For this path not to exist in D', it has to go through node b. But this implies that b would be a cut vertex in D. If b were a cut vertex in D, then there would be a *minimal* cut in D containing exactly a proper subset of the arcs incident to b (consider the set of edges going from b to one of the components of D after the removal of b). A minimal cut in D corresponds to a cycle in P [34, p. 143, ex. 9.2.3]. This implies that there would be a cycle in P whose edges are a proper subset of the edges of H, a contradiction (a proper subset of the edges of any cycle induces an acyclic graph, since it is enough to remove one edge of a cycle to be left with a path, which is acyclic).

Therefore D' is, in fact, a tree. Recall that all nodes of D but b have degree three. Before removing b, we subdivided all of the arcs incident to b. The new nodes have degree one in D', and all others have the same degree as in D, that is, three. ■

A *special 3-coloring of* D' is a partition of the set of nodes of D' into three sets $\{X_1, X_2, X_3\}$ (each set referred to as a *color class*) such that

(1) adjacent nodes have different colors, and
(2) for $i = 1, 2, 3$, if we remove all nodes of color i, in the resulting graph there is a path from any node to a leaf in D'.

Lemma 56.5

There is a special 3-coloring of D'.

Proof

Root D' at one of its leaves. In the rooted D', all internal nodes except the root have two children. Color the root and its unique child with distinct colors. Now, start at level $i = 1$. If there are nodes in level $i + 1$, for each node in level i with children, give its children distinct colors that differ from its own color. Proceed to level $i + 1$.

Clearly, adjacent nodes get distinct colors.

Suppose we remove all nodes of color i, for some i. Let j and k be the two remaining colors. Consider a remaining node. Either it is a leaf in D', and there is nothing to prove, or it is an internal node of the rooted tree D' different from the root (since the root is a leaf of D'). Any internal node colored j that is not the root has a child of color k, and vice versa. This means that there is a path from any node to a leaf in D'. ■

Given a special 3-coloring $\{X_1, X_2, X_3\}$ of D', we now describe three triangular structures S_1, S_2, and S_3 as required by Lemma 56.3. Establish the natural one-to-one correspondence between edges of P and arcs of D': each edge of P corresponds to the unique arc of D' that "crosses" it.

Let S_i be the set of edges in P whose corresponding arc in D' has an endpoint of color i (i.e., in X_i). (An arc has either one or zero endpoints of a given color.)

Lemma 56.6

S_i *is a triangular structure in* P.

Proof

Suppose that there is a cycle C in S_i that is not a triangle. Cycle C partitions the set of faces of P into two sets $F_0 \ni b$ (outside C) and $F_1 \subseteq F$ (inside C), F being the set of faces of D other than b.

Because C is not a triangle, $|F_1| \geq 2$. (If $|F_1| = 1$, then C would be the boundary of the unique face in $F_1 \subseteq F$, which is a triangle.)

A cycle in P corresponds to a minimal cut in D [34, p. 143, ex. 9.2.3]. So F_1 must induce a connected subgraph of D. Since D' differs from D only in arcs incident to b, F_1 induces the same connected subgraph in D'. Also, F_1 consists only of internal nodes of D'.

Thus F_1 induces a connected subgraph of D' with at least two nodes. Hence, in the given special 3-coloring of D', not all nodes of F_1 can be of color i; there is a node d in F_1 of color $j \neq i$.

All leaves of D' are outside C. Since C is in S_i, each edge of C corresponds to an arc in D' with one endpoint of color i. Removing the nodes of D' of color i hence eliminates all arcs with one endpoint inside C and one outside. Thus, after removing from D' the nodes (faces of P) of color i, there can be no path from node d to a leaf of D'. This is a contradiction to the fact that we have a special 3-coloring. ■

Clearly, S_1, S_2, and S_3 satisfy the statement of Lemma 56.3: each edge of P appears in exactly two of them. ■

Theorem 56.8

Let G be a graph with a nonnegative weight function w on its edges and let S be a maximum weight triangular structure in G. Then $w(S) \geq \frac{2}{3}w(P)$ for any outerplanar subgraph P of G.

Proof

By adding edges possibly not in G, extend P to a maximal outerplanar graph $\overline{P}$. For any edge e in $\overline{P}$ but not in G, let $w(e) = 0$. Clearly $w(\overline{P}) \geq w(P)$.

Let S_1, S_2, S_3 be three triangular structures in $\overline{P}$ as given by Lemma 56.3. Each edge of $\overline{P}$ appears in exactly two of these triangular structures. Then $w(S_1) + w(S_2) + w(S_3) = 2w(\overline{P}) \geq 2w(P)$. Moreover, $w(S) \geq w(S_i)$, $i = 1, 2, 3$. Therefore $2w(P) \leq 3w(S)$, implying that $w(S) \geq \frac{2}{3}w(P)$. ■

As there are outerplanar graphs H_i with $2i$ vertices and $3i - 2$ edges that do not have any triangle (all faces except the outer one having size four), the theorem above is tight. Thus we conclude.

Corollary 56.3

Algorithm B, described in Section 56.3, has approximation ratio of $2/3$ *for the unweighted version of* MAXIMUM-WEIGHT OUTERPLANAR SUBGRAPH. *There is a polynomial-time* $(2/3-\epsilon)$*-approximation algorithm for* MAXIMUM-WEIGHT OUTERPLANAR SUBGRAPH.

56.6 Practical Results and Discussion

Exact algorithms for MAXIMUM PLANAR SUBGRAPH have been proposed. Foulds and Robinson [3] present a branch and bound algorithm successful only on small dense graphs. Cimikowski [35] also proposes a branch and bound algorithm. Jünger and Mutzel [9,36] present a branch-and-cut methodology with promising results on unweighted instances. New facets for this method have been proposed by Hicks [37].

Heuristics without proven approximation ratio have been proposed and analyzed experimentally by Cacceta and Kusumah [38] and Poranen [39,40]. In particular, Poranen [40] proposes modified greedy algorithms that improve on experiments over algorithm A from Section 56.3. Precisely, triangles are added also when two of the three vertices are in the same component of $G[E_1]$ (the third being in another component) if the two vertices are adjacent in E_1. Poranen [40] proves that this modified algorithm also has approximation ratio at least 7/18 and conjectures that it has approximation ratio 4/9. We have examples showing that in fact the 7/18 ratio is tight for his algorithm. Poranen reports that the modified greedy algorithm is a slightly better start for simulated annealing heuristics.

For certain classes of inputs, better algorithms are known. Kühn et al. [41] showed that graphs having large minimum degree contain large planar subgraphs. For example, if the minimum degree is at least $1500\sqrt{n}/\epsilon^2$, then every graph G with $n = |V(G)|$ sufficiently large has a planar subgraph with $(2 - \epsilon)n$ edges. Their proofs can be converted in polynomial-time algorithms.

If the input graph is a weighted clique whose edge weights satisfy the triangle inequality, the output of the algorithm that mimics Berman and Ramaiyer's [22] has approximation ratio at least 3/8 [21].

The related problem of GRAPH THICKNESS (given G, partition the edges of G into as few planar subgraphs as possible) has been proven to be NP-hard [42]. A 3-approximation for GRAPH THICKNESS is trivial, via arboricity as in the proof of Lemma 56.1. Finding better approximation algorithms seems very hard.

We conclude with a discussion on improving the approximation ratio for MAXIMUM PLANAR SUBGRAPH and MAXIMUM-WEIGHT PLANAR SUBGRAPH. First we remark that with the current methods adapted from MINIMUM STEINER TREE, MAXIMUM-WEIGHT PLANAR SUBGRAPH is harder than MAXIMUM PLANAR SUBGRAPH, as opposed to many problems mentioned in Ref. [43].

For MAXIMUM-WEIGHT PLANAR SUBGRAPH, there is still room for improvement with triangular structures. Conjecture 56.1, for instance (or some weaker version of it with a bound greater than $1/3 + 1/72$), would already imply a better ratio for MAXIMUM-WEIGHT PLANAR SUBGRAPH.

A natural approach is to consider structures with larger blocks. For example, what is the approximation ratio of an algorithm that produces a "large" structure whose blocks have up to four vertices? (Note that such structures are planar.) Unfortunately, finding such a structure with the maximum number of edges is NP-hard. (One can reduce packing of triangles in graphs of maximum degree bounded by four to this problem.) The approximation algorithm of Berman and Ramaiyer's [22] and the relative greedy algorithm of Zelikovsky [44] can be adapted to use planar structures with blocks of size at most k, for an arbitrary fixed k, and run in time polynomial in n^k, where n denotes, as usual, the number of vertices of the input graph.

For a graph H, denote by $\text{mts}_k(H)$ the maximum number of edges in a subgraph of H whose blocks have size at most k. Theorem 56.3 states that, if H is planar, $\text{mts}_3(H)/|E(H)| \geq 4/9$. Computing the infimum over planar graphs H of $\text{mts}_k(H)/|E(H)|$ for larger values of k is interesting, but we do not believe the values of these infimums are large enough to allow the algorithms of Berman and Ramaiyer's [22] or the relative greedy algorithm of Zelikovsky [44] to improve the 4/9 approximation ratio.

The equivalent of the infima for MINIMUM STEINER TREE are the Steiner ratios for k-restricted Steiner trees and are well studied [45,46] (see also Chapter 42). Here we have that these infima converge to 1/2 as k goes to ∞; the family of planar graphs consisting of two vertices u and v and many paths of length two from u to v shows that all these infima are at most 1/2. Differently, the Steiner ratios for k-restricted Steiner trees tend to 1 as k goes to ∞.

In other words, we believe that algorithm B is better than possible adaptations of Berman and Ramaiyer's or Zelikovsky's algorithm. This type of situation happens in fact for MINIMUM STEINER TREE, where computing an almost optimal 3-restricted Steiner tree (as done by Prömel and Steger [25]) is better than both Berman and Ramaiyer's [22] and the relative greedy approximation algorithm of Zelikovsky [44].

All known MINIMUM STEINER TREE algorithms with better ratio than Prömel and Steger [25] (the best one being the one by Robins and Zelikovsky [47]) use not only the notion of "gain" (as defined in Ref. [46]), but also the notion of "loss." However, the notion of "loss" does not translate easily to MAXIMUM PLANAR SUBGRAPH.

References

[1] Dyer, M. E., Foulds, L. R., and Frieze, A. M., Analysis of heuristics for finding a maximum weight planar subgraph, *Eur. J. Oper. Res.*, 20, 102, 1985.

[2] Foulds, L. R., *Graph Theory Applications*, Springer, New York, 1992.

[3] Foulds, L. R. and Robinson, D. F., Graph-theoretic heuristics for the plant layout problem, *Int. J. Prod. Res.*, 12, 27, 1978.

[4] Krejcirik, K., Computer aided plant layout, *Comp. Aided Design*, 2, 7, 1969.

[5] Mutzel, P., The Maximum Planar Subgraph Problem, Ph.D. thesis, Universität zu Köln, 1994.

[6] Eades, P., Foulds, L., and Giffin, J., An efficient heuristic for identifying a maximal weight planar subgraph, *Combinatorial Mathematics IX*, Lectures Notes in Mathematics, Vol. 952, Springer, Berlin, 1982, p. 239.

[7] Marek-Sadowska, M., Planarization algorithms for integrated circuits engineering, *Proc. IEEE Int. Symp. on Circuits and Systems*, 1978, p. 919.

[8] Di Battista, G., Eades, P., Tamassia, R., and Tollis, I. G., *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, Englewood, NJ, 1998.

[9] Jünger, M. and Mutzel, P., Maximum planar subgraphs and nice embeddings: practical layout tools, *Algorithmica,* 16 (1), 33, 1996. (special issue on graph drawing.)

[10] Liebers, A., Planarizing graphs—A survey and annotated bibliography, *J. Graph Algorithms and Appli.*, 5 (1), 1, 2001.

[11] Batini, C., Nardelli, E., Talamo, M., and Tamassia, R., A graph-theoretic approach to aesthetic layout of information system diagrams, *Proc. 10th Int. Workshop on Graph Theoretic Concepts in Computer Science*, 1984, p. 9.

[12] Tamassia, R., Di Battista, G., and Batini, C., Automatic graph drawing and readability of diagrams, *IEEE Trans. Syst., Man Cybern.*, 18, 61, 1988.

[13] Chiba, N., Nishizeki, T., and Shirakawa, I., An algorithm for maximal planarization of graphs, *Proc. IEEE Int. Symp. on Circuits and Systems*, 1979, p. 649.

[14] Jayakumar, R., Thulasiraman, K., and Swamy, M. N. S., On maximal planarization of nonplanar graphs, *IEEE Trans. Circuits Syst.*, CAS-33, 843, 1986.

[15] Ozawa, T. and Takahashi, H., A graph-planarization algorithm and its applications to random graphs, *Graph Theory and Algorithms*, Lecture Notes in Computer Science, Vol. 108, Springer, Berlin, 1981, p. 95.

[16] Liu, P. C. and Geldmacher, R. C., On the deletion of nonplanar edges of a graph, *Proc. 10th Southeastern Conf. on Combinatorics, Graph Theory, and Computing*, 1977, p. 727.

[17] Yannakakis, M., Node- and edge-deletion NP-complete problems, *Proc. of STOC*, 1978, p. 253.

[18] Călinescu, G., Fernandes, C. G., Finkler, U., and Karloff, H., A better approximation algorithm for finding planar subgraphs, *J. Algorithms*, 27, 269, 1998.

[19] Faria, L., de Figueiredo, C. M. H., and Mendonça, C. F. X., On the complexity of the approximation of nonplanarity parameters for cubic graphs, *Disc. Appl. Math.*, 141, 119, 2004.

[20] Cimikowski, R., An analysis of heuristics for graph planarization, *J. Inf. Opt. Sci.*, 18 (1), 49, 1997.

[21] Călinescu, G., Fernandes, C. G., Karloff, H., and Zelikovsky, A., A new approximation algorithm for finding heavy planar subgraphs, *Algorithmica*, 36 (2), 179, 2003.

[22] Berman, P. and Ramaiyer, V., Improved approximations for the Steiner tree problem, *J. Algorithms*, 17, 381, 1994.

[23] Camerini, P. M., Galbiati, G., and Maffioli, F., Random pseudo-polynomial algorithms for exact matroid problems, *J. Algorithms*, 13, 258, 1992.

[24] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[25] Prömel, M. J. and Steger, A., A new approximation algorithm for the Steiner tree problem with performance ratio 5/3, *J. Algorithms*, 36, 89, 2000.

[26] Korte, B. and Hausmann, D., An analysis of the greedy heuristic for independence systems, *Ann. Disc. Math.*, 2, 65, 1978.

[27] Chen, B., Matsumoto, M., Wang, J., Zhang, Z., and Zhang, J., A short proof of Nash-Williams' theorem for the arboricity of a graph, *Graphs and Combinatorics*, 10, 27, 1994.

[28] Nash-Williams, C. St. J. A., Decomposition of finite graphs into forests, *J. London Math. Soc.*, 39, 12, 1964.

[29] Harary, F., *Graph Theory*, Addison-Wesley, Reading, MA, 1972.

[30] Gabow, H. N. and Stallmann, M., Efficient algorithms for graphic matroid intersection and parity, in *12th Colloq. on Automata, Language and Programming*, Lecture Notes in Computer Science, Vol. 194, Springer, Berlin, 1985, p. 210.

[31] Szigeti, Z., On the graphic matroid parity problem, *J. Combin. Theor. Ser. B*, 88 (2), 247, 2003.

[32] Raz, D., Personal communication, 1996.

[33] Lovász, L. and Plummer, M. D., *Matching Theory*, Elsevier Science, Amsterdam, 1986.

[34] Bondy, J. A. and Murty, U. S. R., *Graph Theory with Applications*, Macmillan, New York, 1976.

[35] Cimikowski, R., Branch-and-bound techniques for the maximum planar subgraph problem, *Int. J. Comput. Math.*, 53, 135, 1994.

[36] Jünger, M. and Mutzel, P., Solving the maximum weight planar subgraph problem, *Proc. 3rd Integer Programming and Comb. Opt. Conf.*, 1993, p. 479.

[37] Hicks, I. V., New Facets for the Planar Subgraph Polytope, submitted for publication, 2005.

[38] Cacceta, L. and Kusumah, Y. S., A new heuristic for facility layout design, *Proc. of Opt., Techniques and Applications*, 1998, p. 287.

[39] Poranen, T., A simulated annealing algorithm for the maximum planar subgraph problem, *Int. J. Comp. Math.*, 81(5), 555, 2004.

[40] Poranen, T., Two new approximation algorithms for the maximum planar subgraph problem, Acta Cybernetica (to appear).

[41] Kühn, D., Osthus, D., and Taraz, A., Large planar subgraphs in dense graphs, *J. Combin. Theory Ser. B* 95 (2), 263, 2005.

[42] Mansfield, A., Determining the thickness of graphs is NP-Hard, *Math. Proc. Cambridge Philos. Soc.*, 93, 9, 1983.

[43] Crescenzi, P., Silvestri, R., and Trevisan, L., To weight or not to weight: Where is the question? *Proc. 4th Israel Symp. on Theory of Computing and Systems*, 1996, p. 68.

[44] Zelikovsky, A., *Better Approximation Bounds for the Network and Euclidean Steiner Tree Problems*, Department of CS, University of Virginia, CS-96-06, 1996.

[45] Du, D.-Z., Zhang, Y.-J., and Feng, Q., On better heuristic for Euclidean Steiner minimum trees, *Proc. FOCS*, 1991, p. 431.

[46] Zelikovsky, A., An 11/6-approximation algorithm for the network Steiner problem, *Algorithmica*, 9, 463, 1993.

[47] Robins, G. and Zelikovsky, A., Tighter bounds for graph Steiner tree approximation, *SIAM J. Disc. Math.*, 19(1), 122, 2005.

57

Edge-Disjoint Paths and Unsplittable Flow

Stavros G. Kolliopoulos
National and Kapodistrian University of Athens

57.1 Introduction

Finding disjoint paths in graphs is a problem that has attracted considerable attention from at least three perspectives: graph theory, VLSI design, and network routing/flow. The corresponding literature is extensive. In this chapter we limit ourselves mostly to results on offline approximation algorithms for problems on general graphs as influenced from the network flow perspective. Surveys examining the underlying graph theory, combinatorial problems in VLSI, and disjoint paths on special graph classes can be found in Refs. [1–8].

An instance of *disjoint paths* consists of a (directed or undirected) graph $G = (V, E)$ and a multiset $\mathcal{T} = \{(s_i, t_i) : s_i \in V, t_i \in V, \;\; i = 1, \ldots, k\}$ of k source–sink pairs. Any source or sink is called a *terminal.* An element of $\mathcal{T}$ is also called a *commodity.* One seeks a set of edge- (or vertex-)disjoint paths $P_1, P_2, \ldots, P_k$, where P_i is an s_i–t_i path, $i = 1, \ldots, k$. In the case of vertex-disjoint paths we are interested in paths that are internally disjoint, that is, a terminal may appear in more than one pair in $\mathcal{T}$. We abbreviate the edge-disjoint paths problem by EDP. The notation introduced will be used throughout the chapter to refer to an input instance. We will also denote $|V|$ by n and $|E|$ by m for the corresponding graph.

On the basis of whether G is directed or undirected and the edge- or vertex-disjointness condition, one obtains four basic problem versions. The following polynomial-time reductions exist among them. Any undirected problem can be reduced to its directed counterpart by replacing an undirected edge with an appropriate gadget; both reductions maintain planarity (see, e.g., Refs. [9] and [4, Chapter 70] for details). An edge-disjoint problem can be reduced to its vertex-disjoint counterpart by replacing G with its line graph (or digraph as the case may be). Directed vertex-disjoint paths reduce to directed edge-disjoint paths by replacing every vertex with a pair of new vertices connected by an edge. There is no known reduction from a directed to an undirected problem. The reader should bear in mind these transformations throughout the chapter. They can serve for translating approximation guarantees or hardness results from the edge-disjoint to the vertex-disjoint setting and vice versa.

The *unsplittable flow* problem (UFP) is the generalization of EDP , where every edge $e \in E$ has a positive capacity u_e, and every commodity i has a demand $d_i > 0$. The demand from s_i to t_i has to be routed in an unsplittable manner, that is, along a single path from s_i to t_i. For every edge e the total demand routed through that edge should be at most u_e. We will often refer to a capacitated graph as a *network.* In a similar manner a vertex-capacitated generalization of vertex-disjoint paths can be defined. UFP was introduced in the Ph.D. thesis of Kleinberg [8]. Versions of the problem had been studied before though not under the UFP moniker (see, e.g., Refs. [10,11]).

If one relaxes the requirement that every commodity should use exactly one path, one obtains the *multicommodity flow* problem which is well known to be solvable in polynomial time. When all the sources of a multicommodity flow instance coincide at a vertex s and all the sinks at a vertex t, we obtain the classical *maximum flow* problem to which we also refer to as *s–t flow.* The relation between UFP and multicommodity flow is an important one to which we shall return often in this chapter. We will denote a solution to either problem as a flow vector f, defined on the edges or the paths of G as appropriate.

57.1.1 Complexity of Disjoint-Path Problems

For general k all four basic problems are *NP*-complete. The undirected vertex-disjoint paths problem was shown to be *NP*-complete by Knuth in 1974 (see Ref. [12]), via a reduction from SAT, and by Lynch [13]. This implies the *NP*-completeness of directed vertex-disjoint paths and directed edge-disjoint paths. Even et al. [14] showed that both problems remain *NP*-complete on directed acyclic graphs (DAGs). In the same paper the undirected edge-disjoint paths problem was shown *NP*-complete even when $\mathcal{T}$ contains only two distinct pairs of terminals. In the case when $s_1 = s_2 = \cdots = s_k$ all four versions are in P as special cases of maximum flow. For planar graphs Lynch's [13] reduction shows *NP*-completeness for undirected vertex-disjoint paths; Kramer and van Leeuwen [15] show that undirected EDP is *NP*-complete. The *NP*-completeness of the directed planar versions follows.

For fixed k, the directed versions are *NP*-complete even for the case of two pairs with opposing source–sinks, that is, (s, t) and (t, s) [16].[1] Undirected vertex-disjoint paths, and by implication edge-disjoint paths as well, can be solved in polynomial time [19]. This is an outcome of the celebrated project of Robertson and Seymour on graph minors. See Ref. [20] for an informal description of the highly impractical Robertson–Seymour algorithm. It is notable that for fixed k, vertex-disjoint paths, and by consequence EDP, can be solved on DAGs by a fairly simple polynomial-time algorithm [16]. Earlier polynomial-time algorithms for $k = 2$ include the one by Perl and Shiloach [9] on DAGs and the ones derived independently by Seymour [21], Shiloach [22], and Thomassen [23] for vertex-disjoint paths on general undirected graphs.

For planar graphs and fixed k the directed vertex-disjoint path problem is in P [24], while the complexity of the edge-disjoint case is open. When the input graph is a tree, Garg et al. [25] gave a polynomial-time algorithm to maximize the number of pairs that can be connected by edge-disjoint paths. The algorithm extends for vertex-disjoint paths (N. Garg, personal communication, July 2005). By total unimodularity, the EDP maximization problem is polynomial-time solvable on *di-trees* as well, that is, directed graphs in which there is a unique directed path from s_i to t_i, for all i; a reduction to a minimum-cost circulation problem is also possible in this case (cf. Ref. [26]). Reducing directed vertex-disjoint paths to EDP maintains the di-tree property, hence the former problem is in P as well. Observe that directed out- and in-trees are special cases of di-trees.

Our presentation will focus mostly on EDP and its generalization to edge-capacitated UFP. We will switch explicitly to vertex-disjoint paths when necessary. Approximation algorithms for vertex-disjoint paths are typically obtained by modifying appropriately algorithms for the edge-disjoint case.

[1] The *NP*-completeness proof holds for a sparse graph with $m = \Theta(n)$; this observation has consequences for hardness of approximation proofs in Refs. [17,18].

57.1.2 Optimization Versions

Two basic *NP*-hard optimization problems are associated with unsplittable flow and hence with EDP. Given a UFP instance an *unsplittable flow solution* or simply a *routing* is a selection of $k' \leq k$ paths, one each for a subset $\mathcal{T}' \subseteq \mathcal{T}$ of k' commodities. Any routing can be expressed as a flow vector f; the flow f_e through edge e equals the sum of the demands using e. A *feasible* routing is one that respects the capacity constraints. In the *maximum demand* optimization problem one seeks a feasible routing of a subset $\mathcal{T}'$ of commodities such that $\sum_{i \in \mathcal{T}'} d_i$ is maximized. The *congestion* of a routing f is defined as $\max_{e \in E}\{\max\{f_e/u_e, 1\}\}$. Note that the events $f_e < u_e$ and $f_e = u_e$ are equivalent for this definition. In the *minimum congestion* optimization problem one seeks a routing of all k commodities that minimizes the congestion, that is, one seeks the minimum $\lambda \geq 1$ such that all k commodities can be feasibly routed if all the capacities are multiplied by λ. From now on, when we refer to EDP without further qualification we imply the maximum demand version of EDP. Some other objective functions of interest will be defined in Section 57.3.

57.1.3 Main Threads

We present now some of the unifying themes in the literature on approximation algorithms for disjoint-path problems.

LP-rounding algorithms. As mentioned above, multicommodity flow is an efficiently solvable relaxation of EDP. Hence it is not accident that multicommodity flow theory has played such an important part in developing algorithms for disjoint-path problems. This brings us to the standard linear programming formulation for multicommodity flow. Let $\mathcal{P}_i$ denote the set of paths from s_i to t_i. Set $\mathcal{P} := \bigcup_{i=1}^{k} \mathcal{P}_i$. Consider the following linear program (LP) for maximum multicommodity flow (MCF):

$$\begin{aligned}
\text{maximize} \quad & \sum_{P \in \mathcal{P}} f_P & & \text{(LP-MCF)} \\
& \sum_{P \in \mathcal{P}_i} f_P \leq d_i & & \text{for } i = 1, \ldots, k \\
& \sum_{P \in \mathcal{P}\,:\, P \ni e} f_P \leq u_e & & \text{for } e \in E \\
& f_P \geq 0 & & \text{for } P \in \mathcal{P}
\end{aligned}$$

The number of variables in the LP is exponential in the size of the graph. By using flow variables defined on the edges one can write an equivalent LP of polynomial size. We choose to deal with the more elegant flow-path formulation. Observe that adding the constraint $f_P \in \{0, d_i\}, \forall P \in \mathcal{P}_i$, to (LP-MCF) turns it into an exact formulation for maximum demand UFP. A similar LP, corresponding to the *concurrent flow* problem, can be written for minimizing congestion (see, e.g., Ref. [27] for details). We call an LP solution for the optimization problem of interest *fractional.* Several early approximation algorithms for UFP, and more generally integer multicommodity flow, work in two stages. First a fractional solution f is computed. Then f is rounded to an unsplittable solution $\hat{f}$ through procedures of varying intricacy, most commonly by randomized rounding as shown by Raghavan and Thompson [28]. The randomized rounding stage can usually be derandomized using the method of conditional probabilities [29–31]. The derandomization component has gradually become very important in the literature for two reasons. First, through the key work of Srinivasan [32,33] on pessimistic estimators, good deterministic approximation algorithms were designed even in cases where the success probability of the randomized experiment was small (see, e.g., Refs. [34,35] for applications to disjoint paths). Second, in some cases the above two-stage scheme can be implemented rather surprisingly without solving first the LP. Instead one designs directly a suitable Langrangean relaxation algorithm implementing the derandomization part. See the work of Young [36] and Chapter 4 in this volume.

We note that some of the approximation ratios obtained through the LP-rounding method can nowadays be matched (or surpassed) by simple combinatorial algorithms. By combinatorial one usually means algorithms restricted to ordered ring operations as opposed to ordered field ones. Two distinct greedy algorithms for EDP were given by Kleinberg [8] (see also Ref. [37]), and Kolliopoulos and Stein [38] (see also Ref. [39]). Most of the subsequent work on combinatorial algorithms uses these two approaches as a basis. Still the influence of rounding methods on the development of algorithms for disjoint-path problems can hardly be overstated. See Chapters 6 and 7 in this volume for further background on LP-based approximation algorithms.

Approximate max-flow min–multicut theorems. One of the first results on disjoint paths and in fact one of the cornerstones of graph theory is Menger's [40] theorem: an undirected graph is k vertex-connected if and only if there are k vertex-disjoint paths between any two vertices. The edge analog holds as well and the min–max relation behind the theorem has resurfaced in a number of guises, most notably as the max-flow min-cut theorem for s–t flows. Let $G = (V, E)$ be undirected. For $U \subseteq V$, define $\delta(U) := \{\{u, v\} \in E : u \in U \text{ and } v \in V \setminus U\}$. Similarly $dem(U)$ is the sum of all demands over commodities which are separated by the cut $\delta(U)$. A necessary condition for the existence of a feasible fractional solution to (LP-MCF) that satisfies all demands is the *cut condition*:

$$\sum_{e \in \delta(U)} u_e \geq dem(U), \qquad \text{for each } U \subseteq V$$

For s–t flows, the max-flow min-cut theorem [41–43] states that the cut condition is sufficient. For undirected multicommodity flow, Hu [44] showed that the cut condition is sufficient for $k = 2$. It fails in general for $k \geq 3$. For directed multicommodity flows there are simple examples with $k = 2$, for which the directed analog of the cut condition holds but the demands cannot be satisfied fractionally (see, e.g., Ref. [4]). For undirected EDP, already for $k = 2$ the cut condition is not sufficient for a solution to exist [1].

Starting with the seminal work of Leighton and Rao [45], a lot of effort has been spent on establishing approximate multicommodity max-flow min-cut theorems. A *multicut* in an undirected graph $G = (V, E)$ is a subset of edges $F \subseteq E$, such that if all edges in F are deleted none of the pairs (s_i, t_i) $i = 1, \ldots, k$ are in the same connected component of the remaining graph. Garg et al. [46] showed constructively that the minimum multicut is always at most $O(\log k)$ times the MCF and this is existentially tight. See Refs. [27,47,48] for surveys of the many results in this area and their applications to approximation algorithms. Most of this work focused originally on fractional flows. The methods were versatile enough to extend to UFP, typically yielding results that were also obtainable via randomized rounding. See, for example, the discussion on the high-capacity UFP in Section 57.3.1 below. Moreover this body of work contributed significantly to the intellectual climate that spawned, among other currents in approximation algorithms, the renewed interest in disjoint paths. This research also produced increased interest in the efficient solution of multicommodity flow problems via combinatorial approximation schemes, thereby producing fast algorithms for solving disjoint-path relaxations. Such approximation schemes had been first investigated by Shahrokhi and Matula [49]. The running time was significantly improved in Ref. [50] with extensions and refinements following in Refs. [51–53]. Extensions of these methods to general fractional packing/covering problems were first pursued in Refs. [54,55]. A representative sample of subsequent work on fractional multicommodity flow and related problems can be found in Refs. [36,56–59].

Finally, one should acknowledge the influence of the Ph.D. thesis of Kleinberg [8] on solidifying the various strands of work on disjoint-path problems up to the mid-1990s. The results in Ref. [8] gave impetus to new research and the thesis itself is a valuable reference tool for earlier work.

The outline of this chapter is as follows. In Section 57.2 we present hardness of approximation results and (mostly greedy) algorithms for EDP. In Section 57.3 we examine the more general UFP problem, properties of the fractional relaxation, and packing integer programs (PIPs). Finally in Section 57.4 we present results on some variants of the basic problems. Unless mentioned otherwise, all of the approximation algorithms we will describe in the upcoming sections work equally well on directed and undirected graphs.

57.2 Algorithms for Edge-Disjoint Paths

In this section we examine the problem of finding a maximum-size set of edge-disjoint paths, mostly from the perspective of combinatorial algorithms. We defer the discussion of the LP-rounding algorithms and the integrality gaps of the linear relaxations until Section 57.3, where we examine them in the more general context of UFP, similarly for some key results on expander graphs and hardness bounds particular to UFP.

57.2.1 Hardness Results

Guruswami et al. [17] showed that on directed graphs it is *NP*-hard to obtain an $O(n^{1/2-\varepsilon})$-approximation for any fixed $\varepsilon > 0$. They gave a gap-inducing reduction from the two-pair decision problem to EDP on a sparse graph with $\Theta(n)$ edges. Since this EDP problem reduces to a vertex-disjoint path instance on a graph with $N = \Theta(n)$ vertices, we obtain that it is *NP*-hard to approximate vertex-disjoint paths on graphs with N vertices within $O(N^{1/2-\varepsilon})$, for any fixed $\varepsilon > 0$. Ma and Wang [60] showed via the PCP theorem that it is *NP*-hard to approximate EDP on directed graphs within $O(2^{\log^{1-\varepsilon} n})$, even when the graph is acyclic. See Chapter 17 in this volume for background on the PCP theorem and the theory of inapproximability. For the undirected edge-disjoint path problem, Andrews and Zhang [61] showed that there is no $O(\log^{1/3-\varepsilon} n)$-approximation algorithm unless $NP \subseteq ZPTIME(n^{poly\log(n)})$. $ZPTIME(n^{poly\log(n)})$ is the set of languages that have randomized algorithms that always give the correct answer in expected running time $n^{poly\log(n)}$. The lower bound was improved to $\Omega(\log^{1/2-\epsilon} n)$ in Ref. [62], under the same complexity-theoretic assumption. Even when congestion $C > 1$ is allowed, Ref. [62] shows that the maximization version is $\log^{\Omega(1/C)} n$-hard to approximate. EDP on undirected graphs was shown MAX SNP-hard in Ref. [25].

57.2.2 Greedy Algorithms

The first approximation algorithm analyzed in the literature for EDP on general graphs seems to be the online *Bounded Greedy Algorithm (BGA)* in the Ph.D. thesis of Kleinberg [8] (see also Ref. [37]). The algorithm is parameterized by a quantity L. The terminal pairs are examined in one pass. When (s_i, t_i) is considered, check if s_i can be connected to t_i by a path of length at most L. If so, route (s_i, t_i) on such a path P_i. Delete P_i from G and iterate. To simplify the analysis we assume that the last terminal pair is always routed if all the previous pairs have been rejected.

The idea behind the analysis of BGA [8] is very simple but it has influenced later works such as Refs. [38], [63], and [64]. Informally it states that *in any graph there cannot be too many long paths that are edge-disjoint.* In Ref. [8] the algorithm was shown to achieve a $(2L + 1)$-approximation if $L = \max\{diam(G), \sqrt{m}\}$. Several people quickly realized that the analysis can be slightly altered to obtain an $O(\sqrt{m})$-approximation. We provide such an analysis with $L = \sqrt{m}$. The first published $O(\sqrt{m})$-approximation for EDP was given by Srinivasan [34] using LP-rounding methods.

Let $\mathcal{O}$ be a maximum-cardinality set of edge-disjoint paths connecting pairs of $\mathcal{T}$. Let $\mathcal{B}$ be the set of paths output by BGA and $\mathcal{O}_u \subset \mathcal{O}$ the set of paths corresponding to terminal pairs unrouted by the BGA. We have that

$$|\mathcal{O}| - |\mathcal{O}_u| = |\mathcal{O} \setminus \mathcal{O}_u| \leq |\mathcal{B}| \qquad (57.1)$$

One tries to relate $|\mathcal{O}_u|$ to $|\mathcal{B}|$. This is done by observing that a commodity l routed in $\mathcal{O}_u$ was not routed in $\mathcal{B}$ because one of the two things happened: (i) no path of length shorter than L exists or (ii) the existing paths from s_l to t_l were blocked by (intersect on at least one edge with) paths selected earlier in $\mathcal{B}$. The paths in $\mathcal{O}_u$ can thus be partitioned into the two corresponding subsets $\mathcal{O}_1$ and $\mathcal{O}_2$. $\mathcal{O}_1$ contains paths blocked by a path in $\mathcal{B}$ and has size at most $L|\mathcal{B}|$, since the elements of $\mathcal{B}$ are edge-disjoint paths of length at most L. The second set $\mathcal{O}_2 := \mathcal{O}_u \setminus \mathcal{O}_1$, consists of disjoint paths longer than L, hence $|\mathcal{O}_2| < m/L$. Therefore

$$|\mathcal{O}_u| < \frac{m}{L} + L|\mathcal{B}| = \sqrt{m} + \sqrt{m}|\mathcal{B}| \leq 2\sqrt{m}|\mathcal{B}| \qquad (57.2)$$

Adding inequalities (57.1) and (57.2) yields that the BGA is an $O(\sqrt{m})$-approximation algorithm. In Section 57.3.3 below we return to the performance of the BGA on expander graphs.

The astute reader has noticed that the idea used in the analysis above is an old one. It goes back to the blocking flow method of Dinitz [65] for the s–t flow problem as applied to unit-capacity networks by Even and Tarjan [66]. A *blocking flow* is a flow that cannot be augmented without rerouting. The blocking flow method iterates over the residual graph. In every iteration a blocking flow is found over the subgraph of the residual graph that contains the edges on a shortest path from s to t. At the end of an iteration the distance from s to t in the new residual graph can be shown to have increased by at least one. When the distance becomes larger than L, the number of edge-disjoint paths from s to t is $O(\min\{m/L, n^2/L^2\})$ and this bounds also the remaining number of augmentations required by the algorithm [66].

Kolliopoulos and Stein [38] made the connection with the blocking flow idea explicit and proposed the offline Greedy_Path algorithm, from now on called simply *the greedy algorithm.* The motivation behind the greedy algorithm was the following: what amount of residual flow has survived if one is never allowed to reroute the flow sent along shortest paths at a given iteration? In every iteration, greedy picks the unrouted (s_i, t_i) pair such that the length of the shortest path P_i from s_i to t_i is minimized. The pair is routed using P_i. The greedy algorithm is easily seen to achieve an $O(\sqrt{m})$-approximation [38]. Using the BGA notation and analysis from above we obtain the following (see also Ref. [64]):

Lemma 57.1

Consider the restriction of the greedy algorithm that stops as soon as the minimum shortest path length among the unrouted pairs exceeds L. The approximation guarantee is at most $\max\{L, |\mathcal{O}_2|\}$.

The analysis in Ref. [38] used the fact that $|\mathcal{O}_2| \leq m/L$. This was extended by Chekuri and Khanna [64].

Theorem 57.1 (Chekuri and Khanna [64])

Using the notation defined above $|\mathcal{O}_2| = O(n^2/L^2)$ for undirected simple graphs and $|\mathcal{O}_2| = O(n^4/L^4)$ for the directed case.

The theorem, together with Lemma 57.1 and Ref. [38], yields immediately that the greedy algorithm achieves an $O(\min\{\sqrt{m}, n^{2/3}\})$-approximation for undirected EDP and an $O(\min\{\sqrt{m}, n^{4/5}\})$ for directed EDP. Varadarajan and Venkataraman [67] improved the bound for directed graphs to $O(\min\{\sqrt{m}, (n \log n)^{2/3}\})$, again for the greedy algorithm. Interestingly, their argument shows the existence of a cut of size $O((n^2/L^2) \log^2(n/L))$ that separates all terminal pairs (s_i, t_i) lying at distance L or more. This brings us almost full circle back to the Even–Tarjan [66] bound for s–t flows. The latter argument demonstrates the existence of a cut of size $O(n^2/L^2)$ when the source is at distance L or more from the sink. Chekuri and Khanna [64] demonstrate an infinite family of directed and undirected instances on which the approximation ratio achieved by the greedy algorithm is $\Omega(n^{2/3})$. New ideas are thus required to bring the approximation down to $O(\sqrt{n})$, which in Ref. [64] is conjectured to be possible. Chekuri et al. [104] and independently Nguyen [105] have recently obtained $O(\sqrt{n})$-approximation algorithms for EDP on undirected graphs and DAGs.

We now sketch the proof of Theorem 57.1 for the undirected case as given by Chekuri and Khanna [64]. The theorem holds for the fractional solution as well, that is, the value ν of the maximum fractional multicommodity flow connecting terminals at distance more than L. Call a vertex of G *high-degree* if its degree is more than $6n/L$ and *low-degree* otherwise. The total capacity incident to low-degree vertices is $O(n^2/L)$. We claim that every s_i–t_i path, $(s_i, t_i) \in \mathcal{T}$, must contain at least $L/6$ of the low-degree vertices. Therefore ν, the sum of flow values over the paths used in the fractional solution, is $O(n^2/L^2)$. To prove the claim consider a breadth-first search tree rooted at s_i and let *layer* L_j be the set of vertices at distance j from s_i. We will show something stronger: there are at least $L/6$ layers among the first L consisting only of low-degree vertices. Partition the layers into blocks of three contiguous layers and let B_j denote the block made up of layers L_{3j+1}, L_{3j+2}, L_{3j+3}. Discard the blocks which contain at least one layer consisting entirely of low-degree vertices. If $L/6$ or more blocks are discarded, we are done. So assume that we are left with at least $L/6$ blocks. The blocks are disjoint, so at least one of the remaining blocks, call it B_*, must contain at most $6n/L$ vertices. Consider a high-degree vertex in the middle layer of B_*. By the breadth-first search property all its neighbors must be within B_* itself, a contradiction. This completes the proof of Theorem 57.1.

Other guarantees for general graphs. In the original paper on the greedy algorithm, it was shown to output a solution of size $\Omega(\max\{OPT^2/m_0, OPT/d_0\})$, where OPT is the optimum, m_0 the minimum number of edges used in an optimal solution, and d_0 the minimum average length of the paths in an optimal solution [38]. The second bound is a straightforward consequence of the first. The first bound is obtained through a somewhat more sophisticated charging scheme for the number of paths in an optimal solution blocked by the paths in $\mathcal{B}$. In conclusion, greedy gives better results in the case where there is a "sparse" optimal solution.

57.2.3 Acyclic Digraphs

The author observed in Ref. [39] that greedy achieves an $o(n)$-approximation if the terminal pairs are disjoint and there is an acyclic optimal solution. In particular, one can show using a result in Ref. [68] that in this case $m_0 = O(n^{3/2})$; an $O(n^{3/4})$-approximation follows. Chekuri and Khanna [64] provided an $O(\sqrt{n}\log n)$-algorithm for DAGs. The following applies to general graphs. Because of the d_0-approximation outlined earlier, one can assume without loss of generality that all shortest s_i–t_i paths have length $\Omega(\sqrt{n})$. Then a counting argument shows that there is a vertex u such that at least $\Omega(OPT/\sqrt{n})$ paths in the optimal solution go through this "congested" vertex u. We guess u and concentrate on finding the maximum-size *u-solution,* to our original EDP instance: this consists only of paths going through u. Devising an $O(\log n)$-approximation algorithm of the LP-rounding variety for this special case gives the desired result. Recently, Nguyen [105] showed that an optimal u-solution is polynomial-time computable in DAGs and undirected graphs.

57.2.4 Vertex-Disjoint Paths

The greedy algorithm, with the obvious modifications, connects a set of terminal pairs of size $\Omega(\max\{OPT/\sqrt{n_0}, OPT^2/n_0, OPT/d_0\})$ [38]. Here n_0 denotes the minimum size of a set of vertices used in the optimal solution and d_0 the minimum average path length in an optimal solution. By the hardness result of Ref. [17] this result is essentially tight on directed graphs, unless $P = NP$.

57.3 The General Unsplittable Flow Problem

We start with some additional definitions. We assume that a UFP instance always satisfies the *balance* (also called *no-bottleneck*) *condition*: $d_{max} := \max_{i=1,\dots,k} d_i \leq u_{min} := \min_{e\in E} u_e$, that is, any commodity can be routed through any of the edges. This assumption is common in the literature and we will refer explicitly to an *extended* UFP instance when the balance condition is not met. In the *weighted* UFP, commodity i has an associated weight (profit) $w_i > 0$; one wants to route feasibly a subset of commodities with maximum total weight. Note that maximizing demand reduces to maximizing the weight: simply set $w_i := d_i$, $i = 1, \dots, k$. Another objective function of interest in addition to maximizing demand and minimizing congestion is routing in the *minimum number of rounds.* A round corresponds to a set of commodities that can be routed feasibly, hence one seeks a minimum-size partition of the set of commodities into feasible unsplittable flow solutions. A *uniform capacity unsplittable flow problem* (UCUFP) is a UFP in which all edges of the input graph have the same capacity value.

57.3.1 Randomized Rounding and UFP

Some of the approximation ratios achieved by LP-rounding that we are about to present are currently also obtainable with simple greedy algorithms. See Section 57.3.3 below. Nevertheless, LP-rounding algorithms are analyzed with respect to the existentially weak optima of the linear relaxations. In addition, their analysis yields upper bounds on the respective integrality gaps. An implementation study comparing the actual performance of the LP-based versus the more combinatorially flavored algorithms would be of interest. For an in-depth survey of randomization for routing problems, see Ref. [69].

Minimizing congestion. The best known algorithm for congestion is also perhaps the best known example of the randomized rounding method of Raghavan and Thompson [28]. A fractional solution f to the concurrent flow problem is computed and then one path is selected independently for every commodity from the following distribution: commodity i is assigned to path $P \in \mathcal{P}_i$ with probability f_P/d_i. An application of the Chernoff [70] bound shows that with high probability the resulting congestion is $O(\log n/\log\log n)$ times the fractional optimum. The process can be derandomized using the method of conditional probabilities [31]. Young [36] shows how to construct the derandomized algorithm without having first obtained the fractional solution.

The analysis of the performance guarantee cannot be improved. Leighton et al. [71] provide an instance on a directed graph on which a fractional solution routes at most $1/\log^c n$ flow per edge, for any constant $c > 0$, while any unsplittable solution incurs congestion $\Omega(\log n/\log\log n)$. If the unsplittable solution uses only paths with nonzero fractional flow the lower bound holds for both undirected and directed instances with optimal UFP congestion 1 [71,72]. Chuzhoy and Naor [73] show that for directed graphs there is no $c \log\log n$-approximation for some constant c, unless $NP \subseteq DTIME(n^{O(\log\log\log n)})$. For undirected graphs, Andrews and Zhang [74] show that congestion cannot be approximated within $(\log\log m)^{1-\varepsilon}$, for any constant $\varepsilon > 0$, unless $NP \subseteq ZPTIME(n^{poly\log(n)})$. Trivially, it is NP-hard to approximate congestion within better than 2 in the case of EDP; this would solve the decision problem.

Maximum demand. Srinivasan published the first $O(\sqrt{m})$-approximation for EDP and more generally UCUFP in Ref. [34]. The first nontrivial $O(\sqrt{m}\log m)$-approximation for UFP was published in the IPCO version of Ref. [38]. Simultaneously and independently, Baveja and Srinivasan refined the results in Ref. [34] to obtain an $O(\sqrt{m})$-approximation for the general UFP; this work was published in Ref. [35]. The Baveja–Srinivasan methods extend the earlier key work of Srinivasan [32,33] on LP-rounding methods for approximating PIPs. We now outline some of the ideas in Refs. [32,34,35]. The algorithm computes first a fractional solution f to the (LP-MCF) linear relaxation (cf. Section 57.1.3). The rounding method has two phases. First, a randomized rounding experiment is analyzed to show that it produces with positive probability a near-optimal feasible unsplittable solution. Second, the experiment is derandomized yielding a deterministic polynomial-time algorithm for computing a feasible near-optimal solution. Let y_* be the fractional optimum.

One starts by scaling down every variable f_P by an appropriate parameter $\alpha > 1$. This is done to boost the probability that after randomized rounding all edge capacities are met. Let B_i denote the event that in the unsplittable solution, the capacity of the edge $e_i \in E$ is violated. Let B_{m+1} denote the event that the routed demand will be less than $y_*/(\beta\alpha)$, for some $\beta > 1$. The quantity $\beta\alpha$ is the targeted approximation ratio. The randomized rounding method of Raghavan and Thompson [28] in the context of UFP works by bounding the probability of the "bad" event $\bigcup_{i=1}^{m+1} B_i$ by $\sum_{i=1}^{m+1} Pr(B_i)$. Srinivasan [34] and later Srinivasan and Baveja [35] exploit the fact that the events $\overline{B_i}$ are *positively correlated*: if it is given that a routing respects the capacities of the edges in some $S \subset E$, the conditional probability that for $e_i \in E \setminus S$, $\overline{B_i}$ occurs, is at least $Pr(\overline{B_i})$. Mathematically this is expressed via the FKG inequality due to Fortuin, Ginibre and Kasteleyn (see Ref. [75, Chapter 6]). Using the positive correlation property, Baveja and Srinivasan [35] obtain a better upper bound on $Pr(\bigcup_{e_i \in E} B_i)$ than the naive union bound and, therefore, can prove the existence of an unsplittable solution while using a better, that is, smaller, $\beta\alpha$ scaling factor than traditional randomized rounding. The second ingredient of Srinivasan's method in Refs. [32,33] is to design an appropriate pessimistic estimator to constructively derandomize the method. Such an estimator is shown for UFP as well in Ref. [35]. The by-now standard derandomization approach of Raghavan [31] fails since it relies precisely on the probability $Pr(\bigcup_{i=1}^{m+1} B_i)$ being upper-bounded by $\sum_{i=1}^{m+1} Pr(B_i)$.

Let d denote the *dilation* of the optimal fractional solution f, that is, the maximum number of edges on any flow-carrying path. The Baveja–Srinivasan algorithm computes a solution to weighted UFP of value

$$\Omega(\max\{(y_*)^2/m,\ y_*/\sqrt{m},\ y_*/d\}) \tag{57.3}$$

The corresponding upper bounds on the integrality gap of (LP-MCF) follow. The analysis of Ref. [32] was simplified by Srinivasan in Ref. [76] by using randomized rounding followed by alteration. Here the outcome of the random experiment is allowed to violate some constraints. It is then altered in a greedy manner to achieve feasibility. The problem-dependent alteration step should be analyzed to quantify the potential degradation of the performance guarantee. This method was applied to UFP in Ref. [77].

For weighted **vertex-disjoint** paths the corresponding bounds hold with n in place of m [35,38]. In addition to the upper bounds on the integrality gap of (LP-MCF) given by Eq. (57.3), the integrality gap for EDP is $O(\sqrt{n})$ on undirected graphs [104] and $O(n^{4/5})$ on directed graphs [64]. The gap is known to be at least $k/2$ by an example in a grid-like planar graph with $k = \Theta(\sqrt{n})$, even for the EDP case [25].

Minimizing the number of rounds. Aumann and Rabani [78] (see also Ref. [8]) show that a ρ-approximation for maximum demand translates to an $O(\rho \log n)$ guarantee for the number of rounds objective. Ref. [35] provides improvements when all edge capacities are unit. Let $\chi(\mathcal{T})$ be the minimum number of rounds. In deterministic polynomial time one can feasibly "route in rounds," the number of rounds being the minimum of (i) $O(\chi(\mathcal{T})d^{\delta} \log n + d(y_* + \log n))$ for any fixed $\delta \in (0, 1)$, (ii) $O(\eta^{-1}d(y_* + \log n))$, if for all i, $d_i \geq \eta$, and (iii) $O\left(\chi(\mathcal{T})\sqrt{m\left(1 + (\log n)/\chi(\mathcal{T})\right)}\right)$ [35]. Minimizing the number of rounds for UFP is related to wavelength assignment in optical networks. Connections routed in the same round can be viewed as being assigned the same wavelength. There is a burgeoning literature dealing with *path coloring* as this problem is often called; usually the focus is on special graph classes. See Ref. [79, Chapter 2] for an introduction to this area.

The high-capacity case. In the *high-capacity* UFP, the minimum-edge capacity is $\Omega(\log m)$ times the maximum demand. An optimal deterministic $O(\log n)$-competitive online algorithm was obtained by Awerbuch et al. [11]. It maintains length functions for the edges that are exponential in the current load. This idea was introduced for multicommodity flow in Ref. [49] and heavily used thereafter (see, e.g., Refs. [36,51,55,56]). Raghavan [31] showed that standard randomized rounding achieves with high probability an $O(1)$-approximation for maximum weight with respect to the fractional optimum. Similarly, one obtains that the high-capacity UFP admits an $O(1)$-approximation for congestion. In general, if $d_{max} \leq u_{min}/B$, for some $B > 1$, various improved bounds that depend on B exist, some obtainable via combinatorial algorithms (see Refs. [18,35,38,63,77] for details. Some particularly good results have been obtained for the *half-disjoint* case, that is, when $B = 2$ [80,81,105].

57.3.2 Packing Integer Programs and UFP

Given $A \in [0, 1]^{M \times N}$, $b \in [1, \infty)^M$, and $c \in [0, 1]^N$ with $\max_j c_j = 1$, a *PIP*, $\mathcal{P} = (A, b, c)$ seeks to maximize $c^T x$ subject to $x \in Z_+^N$ and $Ax \leq b$. Constraints of the form $0 \leq x_j \leq d_j$ are clearly allowed. Let B and ζ denote, respectively, $\min_i b_i$ and the maximum number of nonzero entries in any column of A. The restrictions on the values in A, b, c are without loss of generality; arbitrary nonnegative values can be scaled appropriately [32]. When $A \in \{0, 1\}^{M \times N}$, we say that we have a (0, 1)-PIP. The best guarantees known for PIPs are due to Srinivasan [32,33]; those for (0, 1)-PIPs are better than those known for general PIPs by as much as an $\Omega(\sqrt{M})$ factor.

As witnessed by the (LP-MCF) relaxation, UFP is a packing problem, albeit one with an exponential number of variables. Motivated by UFP, Ref. [38] defined the class of *column-restricted PIPs* (*CPIPs*): these are the PIPs in which all nonzero entries of column j of A have the same value ρ_j, for all j. Observe that a CPIP generalizes Knapsack. If one obtains the fractional solution f to the (LP-MCF) relaxation, one can formulate the *rounding problem* as a polynomial-size CPIP, where the columns of A correspond to the paths used in the fractional solution and the rows to the edges in the graph, hence to capacity constraints. The column value ρ_j equals the demand d_j of the commodity corresponding to the path represented by the column. A preprocessing step requires to transform first the fractional solution to a *fractional single-path solution*. This is a fractional solution in which (i) at most one path per commodity is used and (ii) if a commodity is routed at least a $\Omega(1/\log m)$ fraction of the demand is sent to the sink [38]. In combination with improved bounds for CPIPs this approach yielded the $O(\sqrt{m}\log m)$-approximation for UFP mentioned above. The fractional single-path solution concept resurfaced in the algorithm for EDP on DAGs in Ref. [64] (cf. Section 57.2.3).

A result of independent interest in Ref. [38] shows that any family of column-restricted PIPs can be approximated asymptotically as well as the corresponding family of (0,1)-PIPs. This result is obtained constructively via the *grouping-and-scaling* technique which first appeared in Ref. [82] in the context of single-source UFP (see Section 57.3.4 below). Let z_* be the fractional optimum. For a general CPIP the result of Ref. [38] translates to the existence of an integral solution of value $\Omega\left(\max\left\{\frac{z_*}{M^{1/(\lfloor B\rfloor+1)}}, \frac{z_*}{\zeta^{1/\lfloor B\rfloor}}, z_*\left(\frac{z_*}{M\log\log M}\right)^{1/\lfloor B\rfloor}\right\}\right)$. Baveja and Srinivasan [83] improved the dilation bound for column-restricted PIPs to $\Omega(\frac{z_*}{t^{1/\lfloor B\rfloor}})$, where $t \leq \zeta$ is the maximum column sum of A.

57.3.3 Combinatorial Algorithms and Other Results

For extended UFP with polynomially bounded demands, Ref. [17] gave a simple randomized algorithm that achieves an $O(\sqrt{m}\log^{3/2} m)$-approximation and generalized the greedy algorithm for EDP [38] (cf. Section 57.2.2) to UFP, to obtain an $O(\sqrt{m}\log^2 m)$-approximation. Azar and Regev [18] provided the first strongly polynomial algorithm for weighted UFP that achieves an $O(\sqrt{m})$-approximation. For weighted extended UFP they obtained a strongly polynomial $O(\sqrt{m}\log(2+\frac{d_{max}}{u_{min}}))$-approximation algorithm. By a reduction from the two-pair decision problem, Ref. [18] shows it is *NP*-hard to obtain an $O(n^{1-\varepsilon})$-approximation for extended weighted UFP, for any fixed $\varepsilon > 0$. The lower bound applies with all the commodities sharing the same source but with weights different from the demands. For extended UFP the integrality gap of (LP-MCF) is $\Omega(n)$ even when the input graph is a path [77].

Further progress in terms of greedy algorithms was achieved by Kolman and Scheideler [63] and Kolman [84]. Recall the BGA algorithm from Section 57.2.2. Kolman and Scheideler proposed the *careful BGA*, parameterized by L. The commodities are ordered according to their demands, starting with the largest. Commodity i is accepted if there is a feasible path P for it such that, after routing i, the total flow is larger than half their capacity on at most L edges of P. Let $\mathcal{B}_1$ be the solution thus obtained and $\mathcal{B}_2$ the solution consisting simply of the largest demand routed on any path. The output is $\mathcal{B} := \max\{\mathcal{B}_1, \mathcal{B}_2\}$. In Ref. [63] the careful BGA is shown to achieve an $O(\sqrt{m})$-approximation for extended UFP. Generalizing Theorem 57.1 above to UFP, Kolman [84] showed that the careful BGA achieves an $O(\min\{\sqrt{m}, n^{2/3}\})$-approximation on undirected networks and $O(\min\{\sqrt{m}, n^{4/5}\})$-approximation on directed networks, even for extended UFP. Currently these are the best published bounds for UFP; previously they had been shown for UCUFP in Ref. [64]. Recently, Chekuri et al. [104] obtained an LP-based $O(\sqrt{n})$-approximation algorithm for UFP on undirected graphs.

Guarantees depending on the network structure. Existing approximation guarantees for UFP are rather weak and on directed graphs one cannot hope for significant improvements, unless $P = NP$. A different line of work has aimed for approximation ratios depending on parameters other than n and m. This type of work was originally motivated, in part, by popular hypercube-derived interconnection networks (cf. Ref. [85]). Theoretical advances on these networks are typically facilitated by their rich expansion properties. A graph $G = (V, E)$ is an α-expander if for every set X of at most half the vertices, the number of edges leaving X is at least $\alpha|X|$. Concluding a long line of research, Frieze [86] showed that in any r-regular graph with sufficiently strong expansion properties and r a sufficiently large constant, *any* $\Omega(n/\log n)$ vertex pairs can be connected via edge-disjoint paths. See Ref. [86] for references on the long history of the topic and the precise underlying assumptions. In such an expander the median distance between pairs of vertices is $O(\log n)$, hence the result of Frieze is within a constant factor of optimal. This basic property, that expanders are rich in short edge-disjoint paths, has been exploited in various guises in the literature. Results for fractional multicommodity flows along short paths were first given by Leighton and Rao [45].

Kleinberg and Rubinfeld analyzed the BGA on expanders in Ref. [87]. In the light of Frieze's above result, the BGA achieves an $O(\log n)$-approximation. In Ref. [87] it was also shown that for UCUFP one can efficiently compute a fractional solution that routes at least half the maximum demand with dilation $d = O(\Delta^2\alpha^{-2}\log^3 n)$. Here Δ denotes the maximum degree of the (arbitrary) input graph. The bound on d was improved in Ref. [63]. Kolman and Scheideler introduced a new network measure, the *flow number* $F_{G,u}$, and showed that there is always a near-optimal fractional flow of dilation $O(F_{G,u})$. The flow number is a quantity computable in polynomial time which is defined based on the solution to a multicommodity flow problem on G. If $u_{min} \geq 1$, $F_{G,u}$ is always $\Omega(\alpha^{-1})$ and $O(\Delta\alpha^{-1}\log n)$ [63]. The BGA examining the demands in nonincreasing order and with $L := 4F_{G,u}$ achieves an $O(F_{G,u})$-approximation for UFP [63]. Chakrabarti et al. [77] provide an $O(F_G \log n)$-approximation for UFP where F_G is a definition of the flow number concept of Ref. [63] made independent of capacities. F_G and $F_{G,u}$ coincide on uniform capacity networks. Notably, Ref. [77] presents an $O(\sqrt{\Delta \log n})$-approximation for UCUFP on Δ-regular graphs with sufficiently strong, in the sense of Ref. [86], expansion properties.

57.3.4 Single-Source Unsplittable Flow

Much better approximation guarantees exist for the case where all commodities share the same source, the so-called *single-source* UFP (SUFP). In contrast to single-source EDP, SUFP is strongly *NP*-complete [88]. The version of SUFP with costs has also been studied. In the latter problem every edge $e \in E$ has a nonnegative cost c_e. The cost of an unsplittable flow solution is $\sum_{e\in E} c_e f_e$.

The first constant-factor approximations for all the three main objectives (minimizing congestion, maximizing demand, and minimizing the number of rounds) were given by Kleinberg [88]. The factors were improved by Kolliopoulos and Stein in Ref. [82], where also the first approximations for extended SUFP were given. The grouping-and-scaling technique of Ref. [82] consists of partitioning the original problem into a collection of independent subproblems, each of them with demands in a specified range. The fractional solution is then used to assign capacities to each subproblem. The technique is, in general, useful for translating within constant factors integrality gaps obtained for unit demand instances to arbitrary demand instances. It found further applications, for example, in approximating CPIPs [38,83] (cf. Section 57.3.2 above), in weighted UFP on trees [89], and in Ref. [90]. The currently best constant factors for SUFP were obtained by Dinitz et al. [91], though none of them is known to be best possible under some complexity-class separation assumption. Our understanding seems to be better for congestion. The 2-approximation in Ref. [91] is best possible if the fractional congestion is used as a lower bound. No ratio better than 3/2 is possible unless $P = NP$. The lower bound comes from minimizing makespan on parallel machines with allocation restriction [92], which reduces in an approximation-preserving manner to SUFP. The mentioned scheduling problem is also a special case of the generalized assignment problem for which a simultaneous (2, 1)-approximation for makespan and assignment cost exists [93]. Naturally one wonders whether a simultaneous (2, 1)-approximation for congestion and cost is possible for SUFP. This is an outstanding open problem. The currently best trade-off is a (3, 1)-approximation algorithm due to Skutella [94], which cleverly builds on the earlier (3, 2)-approximation in Ref. [82]. Erlebach and Hall [95] show that it is *NP*-hard to obtain a $(2 - \varepsilon, 1)$-approximation, for any fixed $\varepsilon > 0$. Experimental evaluations of algorithms for congestion can be found in Refs. [96,97].

57.4 Variants of the Basic Problems

In this section we examine some variants of the basic problems. In the *bounded-length* EDP (BLEDP), an additional input parameter M is specified. One seeks a maximum-cardinality set of disjoint s_i–t_i paths, $i = 1, \ldots, k$, under the constraint that the length of each path is at most M. In (s, t)-BLEDP all the pairs share the same source s and sink t. Cases that used to be tractable become *NP*-hard with the length constraint. Both in the vertex and the edge-disjoint case, (s, t)-BLEDP is *NP*-complete on undirected graphs even when M is fixed [98]. For variable M and fixed k, the problems remain *NP*-complete [99]. It is *NP*-hard to approximate (s, t)-BLEDP within $O(n^{1/2-\varepsilon})$ on directed graphs and, unless $NP = ZPP$, BLEDP cannot be approximated in polynomial time within $O(n^{1/2-\varepsilon})$ on undirected graphs, for any fixed $\varepsilon > 0$ [17]. On the positive side it is easy to obtain an $O(\sqrt{m})$-approximation for BLEDP. For the paths in the optimal solution with length at most $M' := \min\{\sqrt{m}, M\}$ the BGA with parameter $L = M'$ achieves an $O(M')$-approximation. This is because, in the notation of Section 57.2.2, $\mathcal{O}_2$ is empty. In contrast, there are at most $\sqrt{m}$ edge-disjoint paths of length more than $\sqrt{m}$. See Ref. [17] for other algorithmic results.

In transportation logistics a commodity may be splittable in different containers, each of them to be routed along a single path. One wishes to bound the number of containers used. This motivates the *b-splittable flow problem*, a relaxed version of UFP where a commodity can be split along *at most* $b \geq 1$ paths, b an input parameter. This problem was introduced and first studied by Baier et al. [100]. Clearly for $b = m$, it reduces to solving the fractional relaxation; it is *NP*-complete for $b = 2$. See Refs. [72,101] for a continuation of the work in Ref. [100]. The author observes in Ref. [102] that the single-source 2-splittable flow problem admits a simultaneous (2, 1)-approximation for congestion and cost. Finally, a problem in a sense complementary to b-splittable flow and with more history is the *multiroute flow*, where for reliability purposes the flow *has to* be split along a given number of edge-disjoint paths. See Ref. [103] for definitions and background.

Acknowledgments

Thanks to Chandra Chekuri, Sanjeev Khanna, Maren Martens, Martin Skutella, and Cliff Stein for valuable comments and suggestions. Thanks to Naveen Garg for a clarification on Ref. [25], to Aris Pagourtzis for pointing out [79], and to Lex Schrijver for information on EDP on planar graphs.

References

[1] Frank, A., Packing paths, cuts and circuits—a survey, in *Paths, Flows and VLSI-Layout*, Korte, B., Lovász, L., Prömel, H. J., and Schrijver, A., Eds., Springer, Berlin, 1990.

[2] Frank, A., Connectivity and network flows, in *Handbook of Combinatorics*, Graham, R., Grötschel, M., and Lovász, L., Eds., Elsevier, Amsterdam, 1995.

[3] Schrijver, A., Homotopic routing methods, in *Paths, Flows and VLSI-Layout*, Korte, B., Lovász, L., Prömel, H. J., and Schrijver, A., Eds., 1990.

[4] Schrijver, A., *Combinatorial Optimization: Polyhedra and efficiency*, Springer, Berlin, 2003.

[5] Möhring, R. H., Wagner, D., and Wagner, F., VLSI network design, in *Handbooks in Operations Research and Management Science, 8: Network Routing*, Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., Eds., 1995, p. 625.

[6] Ripphausen-Lipa, H., Wagner, D., and Weihe, K., Survey on efficient algorithms for disjoint paths problems in planar graphs, in *DIMACS-Series in Discrete Mathematics and Theoretical Computer Science, Vol. 20 on the "Year of Combinatorial Optimization,"* Cook, W., Lovász, L., and Seymour, P. D., Eds., AMS, Providence, RI, 1995, p. 295.

[7] Möhring, R. H. and Wagner, D., Combinatorial topics in VLSI design, annotated bibliography, in *Annotated Bibliographies in Combinatorial Optimization*, Dell'Amico, M., Maffioli, F., and Martello, S., Eds., Wiley, New York, 1997, p. 429.

[8] Kleinberg, J. M., Approximation Algorithms for Disjoint Paths Problems, Ph.D. thesis, MIT, Cambridge, MA, May 1996.

[9] Perl, Y. and Shiloach, Y., Finding two disjoint paths between two pairs of vertices in a graph, *JACM*, 25, 1, 1978.

[10] Cosares, S. and Saniee, I., An optimization problem related to balancing loads on SONET rings, *Telecommun. Syst.*, 3, 165, 1994.

[11] Awerbuch, B., Azar, Y., and Plotkin, S., Throughput-competitive online routing, *Proc. of FOCS,* 1993, p. 32.

[12] Karp, R. M., On the computational complexity of combinatorial problems, *Networks*, 5, 45, 1975.

[13] Lynch, J. F., The equivalence of theorem proving and the interconnection problem, *ACM SIGDA Newslett.*, 5, 31, 1975.

[14] Even, S., Itai, A., and Shamir, A., On the complexity of timetable and multicommodity flow problems, *SIAM J. Comput.*, 5, 691, 1976.

[15] Kramer, M. R. and van Leeuwen, J., The complexity of wire-routing and finding minimum-area layouts for arbitrary VLSI circuits, in *VLSI Theory*, Preparata, F. P., Ed., JAI Press, Greenwich, Connecticut, 1984, p. 129.

[16] Fortune, S., Hopcroft, J., and Wyllie, J., The directed subgraph homeomorphism problem, *Theor. Comput. Sci.*, 10, 111, 1980.

[17] Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., and Yannakakis, M., Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems, *JCSS*, 67, 473, 2003.

[18] Azar, Y. and Regev, O., Strongly polynomial algorithms for the unsplittable flow problem, *Proc. of IPCO*, Lecture Notes in Computer Science, Vol. 2081, 2001, p. 15.

[19] Robertson, N. and Seymour, P. D., Graph Minors XIII. The disjoint paths problem, *J. Combinatorial Theory B*, 63, 65, 1995.

[20] Bienstock, D. and Langston, M. A., Algorithmic implications of the Graph Minor Theorem, in *Handbooks in Operations Research and Management Science, 7: Network models*, Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., Eds., North-Holland, Amsterdam, 1995.

[21] Seymour, P. D., Disjoint paths in graphs, *Discrete Math.*, 29, 293, 1980.

[22] Shiloach, Y., A polynomial solution to the undirected two paths problem, *JACM*, 27, 445, 1980.

[23] Thomassen, C., 2-linked graphs, *Eur. J. Combinatorics*, 1, 371, 1980.

[24] Schrijver, A., Finding *k* disjoint paths in a directed planar graph, *SIAM J. Comput.*, 23, 780, 1994.

[25] Garg, N., Vazirani, V., and Yannakakis, M., Primal-dual approximation algorithms for integral flow and multicut in trees, *Algorithmica*, 18, 3, 1997.

[26] Costa, M. C., Létocart, L., and Roupin, F., A greedy algorithm for multicut and integral multiflow in rooted trees, *Oper. Res. Lett.*, 31, 21, 2003.

[27] Vazirani, V. V., *Approximation Algorithms*, Springer, Berlin, 2001.

[28] Raghavan, P. and Thompson, C. D., Randomized rounding: a technique for provably good algorithms and algorithmic proofs, *Combinatorica*, 7, 365, 1987.

[29] Erdős, P. and Selfridge, J. L., On a combinatorial game, *J. Combinatorial Theor. A*, 14, 298, 1973.

[30] Spencer, J., *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, 1987.

[31] Raghavan, P., Probabilistic construction of deterministic algorithms: approximating packing integer programs, *JCSS*, 37, 130, 1988.

[32] Srinivasan, A., Improved approximation guarantees for packing and covering integer programs, *SIAM J. Comput.*, 29, 648, 1999.

[33] Srinivasan, A., An extension of the Lovász Local Lemma and its applications to integer programming, *Proc. of SODA,* 1996, p. 6.

[34] Srinivasan, A., Improved approximations for edge-disjoint paths, unsplittable flow and related routing problems, *Proc. of FOCS,* 1997, p. 416.

[35] Baveja, A. and Srinivasan, A., Approximation algorithms for disjoint paths and related routing and packing problems, *Math. Oper. Res.*, 25, 255, 2000.

[36] Young, N. E., Randomized rounding without solving the linear program, *Proc. of SODA,* 1995, p. 170.

[37] Kleinberg, J. M. and Tardos, É., Disjoint paths in densely-embedded graphs, *Proc. of FOCS,* 1995, p. 52.

[38] Kolliopoulos, S. G. and Stein, C., Approximating disjoint-path problems using packing integer programs, *Math. Program A*, 99, 63, 2004.

[39] Kolliopoulos, S. G., Exact and Approximation Algorithms for Network Flow and Disjoint-Path Problems, Ph.D. thesis, Dartmouth College, Hanover, NH, 1998.

[40] Menger, K., Zur allgemeinen kurventheorie, *Fundam. Math.*, 10, 96, 1927.

[41] Ford, L. R. and Fulkerson, D. R., Maximal flow through a network, *Can. J. Math.*, 8, 399, 1956.

[42] Fulkerson, D. R. and Dantzig, G. B., Computation of maximum flow in networks, *Naval Res. Logistics Quart.*, 2, 277, 1955.

[43] Elias, P., Feinstein, A., and Shannon, C. E., Note on maximum flow through a network, *IRE Trans. Inf. Theor.*, 117, 1956.

[44] Hu, T. C., Multi-commodity network flows, *Oper. Res.*, 11, 344, 1963.

[45] Leighton, T. and Rao, S., Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms, *JACM*, 46, 787, 1999.

[46] Garg, N., Vazirani, V., and Yannakakis, M., Approximate max-flow min-(multi)cut theorems and their applications, *SIAM J. Comput.*, 25, 235, 1996.

[47] Shmoys, D. B., Cut problems and their applications to Divide and Conquer, in *Approximation Algorithms for NP-Hard Problems*, Hochbaum, D. S., Ed., PWS, Boston, 1997, p. 192.

[48] Costa, M. C., Létocart, L., and Roupin, F., Minimal multicut and maximum integer multiflow: a survey, *Eur. J. Oper. Res.*, 162, 55, 2005.

[49] Shahrokhi, F. and Matula, D. W., The maximum concurrent flow problem, *JACM*, 37, 318, 1990.

[50] Klein, P., Plotkin, S. A., Stein, C., and Tardos, É., Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts, *SIAM J. Comput.*, 23, 466, 1994.

[51] Leighton, T., Makedon, F., Plotkin, S., Stein, C., Tardos, É., and Tragoudas, S., Fast approximation algorithms for multicommodity flow problems, *JCSS*, 50, 228, 1995.

[52] Goldberg, A., A natural randomization strategy for multicommodity flow and related algorithms, *Inf. Process. Lett.*, 42, 249, 1992.

[53] Radzik, T., Fast deterministic approximation for the multicommodity flow problem, *Proc. of SODA,* 1995, p. 486.

[54] Grigoriadis, M. D. and Khachiyan, L. G., Fast approximation schemes for convex programs with many blocks and coupling constraints, *SIAM J. Optimization*, 4(1), 86, 1994.

[55] Plotkin, S., Shmoys, D. B., and Tardos, É., Fast approximation algorithms for fractional packing and covering problems, *Math. Oper. Res.*, 20, 257, 1995.

[56] Garg, N. and Könemann, J., Faster and simpler algorithms for multicommodity flow and other fractional packing problems, *Proc. of FOCS,* 1998, p. 300.

[57] Fleischer, L., Approximating fractional multicommodity flows independent of the number of commodities, *SIAM J. Discrete Math.*, 13, 505, 2000.

[58] Karakostas, G., Faster approximation schemes for fractional multicommodity flow problems, *Proc. of SODA,* 2002, p. 166.

[59] Bienstock, D. and Iyengar, G., Solving fractional packing problems in $O^*(1/\varepsilon)$ iterations, *Proc. of STOC,* 2004, p. 146.

[60] Ma, B. and Wang, L., On the inapproximability of disjoint paths and minimum Steiner forest with bandwidth constraints, *JCSS*, 60, 1, 2000.

[61] Andrews, M. and Zhang, L., Hardness of the undirected edge-disjoint paths problem, *Proc. of STOC,* 2005, p. 276.

[62] Andrews, M., Chuzhoy, J., Khanna, S., and Zhang, L., Hardness of undirected edge disjoint paths with congestion, *Proc. of FOCS,* 226–244, 2005.

[63] Kolman, P. and Scheideler, C., Improved bounds for the unsplittable flow problem, *J. Algorithms*, Vol. 61, 20–44, 2006.

[64] Chekuri, C. and Khanna, S., Edge disjoint paths revisited, *Proc. of SODA,* 2003, p. 628.

[65] Dinitz, E. A., Algorithm for solution of a problem of maximum flow in networks with power estimation, *Soviet Math. Dokl.*, 11, 1277, 1970.

[66] Even, S. and Tarjan, R. E., Network flow and testing graph connectivity, *SIAM J. Comput.*, 4, 507, 1975.

[67] Varadarajan, K. and Venkataraman, G., Graph decomposition and a greedy algorithm for edge-disjoint paths, *Proc. of SODA,* 2004, p. 379.

[68] Karger, D. R. and Levine, M. S., Finding maximum flows in simple undirected graphs seems easier than bipartite matching, *Proc. of STOC,* 1998.

[69] Srinivasan, A., A survey of the role of multicommodity flow and randomization in network design and routing, in *(AMS) Series in Discrete Mathematics and Theoretical Computer Science*, Pardalos, P. M., Rajasekaran, S., and Rolim, J., Eds., American Mathematical Society (AMS), Providence, RI, Vol. 43, 1999, p. 271.

[70] Chernoff, H., A measure of the asymptotic efficiency for tests of a hypothesis based on sum of observations, *Ann. Math. Stat.*, 23, 493, 1952.

[71] Leighton, T., Rao, S., and Srinivasan, A., Multicommodity flow and circuit switching, in *Intl. Conf. on System Sciences*, 1998, p. 459.

[72] Martens, M. and Skutella, M., Flows on few paths: algorithms and lower bounds, *Proc. of ESA*, Lecture Notes in Computer Science, Vol. 3221, 2004, p. 520.

[73] Chuzhoy, J. and Naor, S., New hardness results for congestion minimization and machine scheduling, *Proc. of STOC,* 2004, p. 28.

[74] Andrews, M. and Zhang, L., Hardness of the undirected congestion minimization problem, *Proc. of STOC,* 2005, p. 284.

[75] Alon, N. and Spencer, J., *The Probabilistic Method*, 2nd ed., Wiley, New York, 2000.

[76] Srinivasan, A., New approaches to covering and packing problems, *Proc. of SODA,* 2001, p. 567.

[77] Chakrabarti, A., Chekuri, C., Gupta, A., and Kumar, A., Approximation algorithms for the unsplittable flow problem, *Proc. APPROX*, Lecture Notes in Computer Science, Vol. 2462, 2002, p. 51.

[78] Aumann, Y. and Rabani, Y., Improved bounds for all-optical routing, *Proc. of SODA,* 1995, p. 567.

[79] Stefanakos, S., On the Design and Operation of High-Performance Optical Networks, Ph.D. thesis, ETH Zurich, No. 15691, 2004.

[80] Kleinberg, J. M., Decision algorithms for unsplittable flow and the half-disjoint paths problem, *Proc. of STOC,* 1998, p. 530.

[81] Chekuri, C., Khanna, S., and Shepherd, B., Edge-disjoint paths in planar graphs, *Proc. of FOCS,* 2004, p. 71.

[82] Kolliopoulos, S. G. and Stein, C., Approximation algorithms for single-source unsplittable flow, *SIAM J. Comput.*, 31, 919, 2002.

[83] Baveja, A. and Srinivasan, A., Approximating low-congestion routing and column-restricted packing problems, *Inf. Process Lett.*, 74, 19, 2000.

[84] Kolman, P., A note on the greedy algorithm for the unsplittable flow problem, *Inf. Process Lett.*, 88, 101, 2003.

[85] Scheideler, C., *Universal Routing Strategies for Interconnection Networks*, Lecture Notes in Computer Science, Springer, Berlin, Vol. 1390, 1998.

[86] Frieze, A. M., Edge-disjoint paths in expander graphs, *SIAM J. Comput.*, 30, 1790, 2001.

[87] Kleinberg, J. M. and Rubinfeld, R., Short paths in expander graphs, *Proc. of FOCS,* 1996, p. 86.

[88] Kleinberg, J. M., Single-source unsplittable flow, *Proc. of FOCS,* 1996, p. 68.

[89] Chekuri, C., Mydlarz, M., and Shepherd, F. B., Multicommodity demand flow in a tree, *Proc. of ICALP*, Lecture Notes in Computer Science, Vol. 2719, 2003, p. 410.

[90] Chekuri, C., Khanna, S., and Shepherd, B., Multicommodity flow, well-linked terminals and routing problems, *Proc. of STOC,* 2005, p. 183.

[91] Dinitz, Y., Garg, N., and Goemans, M. X., On the single-source unsplittable flow problem, *Combinatorica*, 19, 1, 1999.

[92] Lenstra, J. K., Shmoys, D. B., and Tardos, É., Approximation algorithms for scheduling unrelated parallel machines, *Math. Program A*, 46, 259, 1990.

[93] Shmoys, D. B. and Tardos, É., An approximation algorithm for the generalized assignment problem, *Math. Program. A*, 62, 461, 1993.

[94] Skutella, M., Approximating the single-source unsplittable min-cost flow problem, *Math. Program B*, 91, 493, 2002.

[95] Erlebach, T. and Hall, A., NP-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow, *J. Scheduling*, 7, 223, 2004.

[96] Kolliopoulos, S. G. and Stein, C., Experimental evaluation of approximation algorithms for single-source unsplittable flow, *Proc. of IPCO*, Lecture Notes in Computer Science, Vol. 1610, 1999, p. 328.

[97] Kolliopoulos, S. G. and Du, J., Implementing approximation algorithms for single-source unsplittable flow, *Proc. of WEA,* Lecture Notes in Computer Science, Vol. 3059, 2004, p. 213.

[98] Itai, A., Perl, Y., and Shiloach, Y., The complexity of finding maximum disjoint paths with length constraints, *Networks*, 12, 277, 1982.

[99] Li, C., McCormick, T., and Simchi-Levi, D., The complexity of finding two disjoint paths with min-max objective function, *Discrete Appl. Math.*, 26, 105, 1990.

[100] Baier, G., Köhler, E., and Skutella, M., On the k-splittable flow problem, *Algorithmica,* 42, 2005, 231–248.

[101] Koch, R., Skutella, M., and Spenke, I., Approximation and complexity of k-splittable flows, *Proc. WAOA*, Springer, Berlin, 2005.

[102] Kolliopoulos, S. G., Minimum-cost single-source 2-splittable flow, *Inf. Process Lett.*, 94, 15, 2005.

[103] Bagchi, A., Chaudhary, A., and Kolman, P., Short length Menger's theorem and reliable optical routing, *Theor. Comput. Sci.*, 339, 315, 2005.

[104] Chekuri, C., Khanna, S., Shepherd, B., An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow, *Theory of Computing*, 2, 137–146, 2006.

[105] Nguyen, T., On the disjoint paths problem, *Operations Research Letters*, 35, 10–16, 2007.

58 Approximating Minimum-Cost Connectivity Problems

Guy Kortsarz
Rutgers University

Zeev Nutov
The Open University of Israel

58.1 Introduction

We survey approximation algorithms and hardness results for versions of the *Generalized Steiner Network* (GSN) problem in which we seek to find a low-cost subgraph (where the cost of a subgraph is the sum of the costs of its edges) that satisfies prescribed connectivity requirements. These problems include the following well-known problems: min-cost k-flow, min-cost spanning tree, traveling salesman, directed/undirected Steiner tree, Steiner forest, k-edge/node-connected spanning subgraph, and others.

The type of problems we consider can be formally defined using the following unified framework. Let $G = (V, E)$ be a (possibly directed) graph and let $S \subseteq V$. The *S-connectivity* $\lambda^S_G(u, v)$ of (u, v) in G is the maximum number of uv-paths such that no two of them have an edge or a node in $S - \{u, v\}$ in common.

Generalized Steiner Network
Instance: A (possibly directed) graph $\mathcal{G} = (V, \mathcal{E})$ with costs $\{c_e : e \in E\}$ on the edges, a node subset $S \subseteq V$, and a nonnegative integer *requirement function* $r(u, v)$ on $V \times V$.
Objective: Find a minimum-cost spanning (i.e., on the same node set) subgraph $G = (V, E)$ of $\mathcal{G}$ so that

$$\lambda^S_G(u, v) \geq r(u, v) \quad \text{for all } (u, v) \in V \times V \tag{58.1}$$

Extensively studied particular choices of S are the *edge-* ($S = \emptyset$), the *node-* ($S = V$), and the *element-* ($r(u, v) = 0$ whenever $u \in S$ or $v \in S$) GSN. For brevity, if $r(u, v)$ is not specified, then $r(u, v) = 0$ by default. We may assume that the input graph $\mathcal{G}$ is complete, by assigning infinite costs to "forbidden" edges. Under this assumption, we categorize the edge costs as follows:

- *Augmentation problems* ($\{0, 1\}$-edge costs). Here we are given a graph G_0, and the goal is to find a min-size augmenting edge set F of new edges (any edge is allowed) so that Eq. (58.1) holds for $G = G_0 + F$.
- *Min-size subgraph problems ($\{1, \infty\}$-edge costs, known also as "uniform costs").* Given a graph $\mathcal{H}$ (formed by the edges of cost 1 of $\mathcal{G}$) find a min-size spanning subgraph G of $\mathcal{H}$ so that Eq. (58.1) holds.
- *Metric costs.* Here we assume that the edge costs satisfy the triangle inequality.
- *General (nonnegative) costs.*

For each type of costs, we consider three types of requirements:

- *Rooted (single source/sink) requirements.* That is, there is $s \in V$ so that if $r(u, v) > 0$, then $u = s$ for directed graphs, and $u = s$ or $v = s$ for undirected graphs.
- *Uniform requirements.* $r(u, v) = k$ for every pair $(u, v) \in V \times V$.
  The corresponding "edge" and "node" versions are the *k-Edge-Connected Spanning Subgraph* (k-ECSS) and the *k-(Node-)Connected Spanning Subgraph* (k-CSS) problems.
- *Arbitrary (nonnegative) requirements.*

In the capacitated GSN, every edge e of $\mathcal{G}$ has a capacity $u(e)$ and the costs are per unit of capacity (the capacitated GSN is reduced in pseudopolynomial time to the uncapacitated GSN by replacing every edge e with $u(e)$ copies of e). For simplicity, we consider the uncapacitated case only, and in addition assume that $r_{\max} = \max_{u,v \in V} r(u, v)$ is bounded in a polynomial in $n = |V|$. However, most algorithms are easily adjusted to get the same performance without these simplifying assumptions.

Many well-known problems are particular cases of GSN. When there is only one pair (u, v) with $r(u, v) > 0$ we get the (uncapacitated) min-cost k-flow problem, which is solvable in polynomial time (cf., Ref. [1]). The undirected 1-ECSS (and 1-CSS) is just the Minimum Spanning Tree problem; however, the directed 1-ECSS (and 1-CSS) is NP-hard. The undirected/directed rooted GSN with $r(u, v) \in \{0, 1\}$ is the extensively studied Undirected/Directed Steiner Tree problem (cf., Refs. [2,3]). The undirected GSN with $r(u, v) \in \{0, 1\}$ is the Steiner Forest problem which admits a 2-approximation algorithm. Several other well-known problems are also particular cases of GSN. In this survey we focus on algorithms for edge and node connectivity and $r_{\max} = \max_{u,v \in V} r(u, v)$ arbitrary, although there are many interesting results for element connectivity [4–8], as well as for small requirements, for example, Refs. [3,6,9–14]. See also a previous survey in Ref. [15]. We survey only *approximation* algorithms (for exact algorithms see Refs. [8,16,17]), with the currently best known approximation ratios. Table 58.1 summarizes the currently best known approximation ratios and hardness results for edge/node-connectivity and uniform/general requirements.

Small Requirements

For node-GSN with $\{0, 1\}$ costs the following results are known. The problem admits an $O(r_{\max} \cdot \log n)$-approximation algorithm [22] (for $S \neq V$ the approximation ratio in Ref. [22] is $O(\log n)$). For $r(u, v) \in \{0, 2\}$ the problem is NP-hard and admits a 3/2-approximation algorithm [32]. For uniform requirements $r(u, v) = k$ for all $u, v \in V$ the complexity status is not known for undirected graphs, but for any fixed k an optimal solution can be computed in polynomial time [33]. For rooted uniform requirements (in undirected graphs) the situation is similar [34]. For $\{1, \infty\}$-costs and uniform requirements the following approximation ratios are known: 5/4 for undirected 2-ECSS [11], 4/3 for undirected 2-CSS [14], and $(\pi^2/6 + \varepsilon)$ for directed 1-CSS [13]. For metric costs, both 2-ECSS and 2-CSS admit a 3/2-approximation algorithm, [10]. For undirected k-CSS with arbitrary costs and $k \leq 8$ there are $k/2$-approximation algorithms [9,12].

TABLE 58.1 Approximation Ratios and Hardness Results for GSN Problems. MC and GC Stand for Metric and General Costs, UR and GR Stand for Uniform and General Requirements, Respectively. $\alpha = \min\left\{\frac{n}{n-k}, \frac{\sqrt{k}}{\ln k}\right\}$.

	Edge-Connectivity		Node-Connectivity	
c & r	Undirected	Directed	Undirected	Directed
$\{0, 1\}$ & UR	in P [18]	in P [19]	$\min\{2, 1 + \frac{k^2}{2\mathsf{opt}}\}$[20,21]	in P [20]
$\{0, 1\}$ & GR	in P [19]	$\Theta(\log n)$ [22]	$\Omega(2^{\log^{1-\varepsilon} n})$ [7]	$\Omega(2^{\log^{1-\varepsilon} n})$ [7]
$\{1, \infty\}$ & UR	$1 + 2/k$ [23,24]	$1 + 2/k$ [23]	$1 + 1/k$ [24]	$1 + 1/k$ [24]
$\{1, \infty\}$ & GR	2 [25]	$\Omega(2^{\log^{1-\varepsilon} n})$ [26,27]	$\Omega(2^{\log^{1-\varepsilon} n})$ [27]	$\Omega(2^{\log^{1-\varepsilon} n})$ [26,27]
MC & UR	2 [28]	2 [28]	$2 + (k-1)/n$ [12]	$2 + k/n$ [12]
MC & GR	2 [25]	$\Omega(2^{\log^{1-\varepsilon} n})$ [26]	$O(\log r_{\max})$ [29]	$\Omega(2^{\log^{1-\varepsilon} n})$ [26]
GC & UR	2 [28]	2 [28]	$O(\log k)$, $n \geq 2k^2$ [30]	$O(\alpha \cdot \log^2 k)$ [31]
GC & GR	2 [25]	$\Omega(2^{\log^{1-\varepsilon} n})$ [26]	$\Omega(2^{\log^{1-\varepsilon} n})$ [27]	$\Omega(2^{\log^{1-\varepsilon} n})$ [26]

Element-Connectivity

Most approximation algorithms for node connectivity can be extended to element-connectivity, but in many cases better approximation ratios are possible. For general requirements and general edge-costs undirected element-GSN admits a 2-approximation algorithm [5,6]. For $\{0, 1\}$ costs the problem is NP-hard (even for $r(u, v) \in \{0, 2\}$). For $\{0, 1\}$ costs the best known approximation ratios are [7]: 7/4 for arbitrary requirements, and 3/2 for $\{0, 1, 2\}$- or $\{0, k\}$-requirements.

58.2 Preliminaries

We now define some notation. An edge from u to v is denoted by uv. A *uv-path* is a path from u to v. For an arbitrary two sets A, B of nodes and edges (or graphs) $A - B$ is the set (or graph) obtained by deleting B from A, where deletion of a node implies also deletion of all the edges incident to it; similarly, $A + B$ is the set (graph) obtained by adding B to A. Let H be a (possibly directed) graph or an edge set on node set V. For disjoint $X, Y \subseteq V$ we denote by $\delta_H(X, Y)$ the set $\{uv \in H : u \in X, v \in Y\}$ of the edges in H from X to Y and $d_H(X, Y) = |\delta_H(X, Y)|$; for brevity, $\delta_H(X) = \delta_H(X, V - X)$ and $d_H(X) = |\delta_H(X)|$. Let $\Gamma_H(X)$ be the set $\{v \in V - X : uv \in H$ for some $u \in X\}$ of *neighbors* of X in H. We sometimes omit the subscripts if they are clear from the context. Given a graph, we call the new edges that can be added to it *links*, to distinguish them from the existing edges. Let opt denote the optimal solution value of an instance at hand.

Proposition 58.1

If there exists a ρ-approximation algorithm for the directed GSN *then there exists a 2ρ-approximation algorithm for the undirected* GSN.

Proof

Given an instance $\mathcal{I} = (\mathcal{G}, S, c, r)$ of an undirected GSN, obtain an instance $\mathcal{I}' = (\mathcal{G}', S, c', r')$ of a directed GSN as follows. Replace every edge uv of $\mathcal{G}$ by the two opposite directed edges uv, vu having the same cost as uv, and for every $u, v \in V$ set $r'(u, v) = r'(v, u) = r(uv)$. Then apply the ρ-approximation algorithm on $\mathcal{I}'$ to compute a subgraph D' of $\mathcal{G}'$, and output its underlying graph G. It is easy to see that G is a feasible solution for $\mathcal{I}$. Furthermore, if H is an arbitrary subgraph of $\mathcal{G}$, and H' is the corresponding subgraph of $\mathcal{G}'$, then $c'(H') = 2c(H)$ and H is a feasible solution for $\mathcal{I}$ if and only if, H' is a feasible for $\mathcal{I}'$. In particular, $\mathsf{opt}' \leq 2\rho \cdot \mathsf{opt}$, where opt and opt' denote the optimal solution value of $\mathcal{I}$ and $\mathcal{I}'$, respectively. Thus $c(G) \leq c'(D') = \rho \cdot \mathsf{opt}' \leq 2\rho \cdot \mathsf{opt}$. □

Proposition 58.1 indicates that undirected problems cannot be much harder to approximate than the directed ones; note that the reduction in the proof is "cost-type preserving."

We now briefly discuss some algorithms for rooted requirements (for additional literature see Refs. [3,17,34–37]). A graph $G = (V, E)$ is said to be *k-outconnected from s* if it contains k-internally disjoint sv-paths for every $v \in V - s$; G is *k-inconnected to s* if it contains k-internally disjoint vs-paths for every $v \in V - s$ (for undirected graphs these two concepts are the same). When the paths are only required to be edge-disjoint, we say that G is *k-edge-outconnected from s* or *k-edge-inconnected to s*, respectively. Particular important cases of the rooted requirements are the *k-Edge-Outconnected Subgraph* (k-EOS) and the *k-Outconnected Subgraph* (k-OS) problems, where $r(s, v) = k$ for every $v \in V$. For *directed* graphs, both k-EOS and k-OS can be solved in polynomial time, see Refs. [35,36], respectively. This implies:

Theorem 58.1

Undirected k-EOS and k-OS admit a 2-approximation algorithm.

Proof

We prove the statement for k-OS; the proof for k-EOS is identical. The algorithm is as follows. Replace every edge uv of $\mathcal{G}$ by the two opposite directed edges uv, vu having the same cost as uv. Then in the obtained directed graph $\mathcal{G}'$ with cost function c' compute an optimal k-outconnected from s spanning subgraph D' of $\mathcal{G}'$, and output its underlying graph G. It is easy to see that G is a feasible solution. Furthermore, if H is an arbitrary subgraph of $\mathcal{G}$, and H' is the corresponding subgraph of $\mathcal{G}'$, then $c'(H') = 2c(H)$ and H is k-outconnected from s if, and only if, H' is k-outconnected from s. In particular, $\mathsf{opt}' \leq 2\rho \cdot \mathsf{opt}$, where opt and opt' denote the optimal solution value to $\mathcal{G}$ and $\mathcal{G}'$, respectively. Thus $c(G) \leq c'(D') = \rho \cdot \mathsf{opt}' \leq 2\rho \cdot \mathsf{opt}$. □

Theorem 58.1 is widely used for designing approximation algorithms for k-ECSS and k-CSS. For example, it was observed in Ref. [28] that (a possibly directed) graph $G = (V, E)$ is k-edge-connected if, and only if, G is both k-edge-outconnected from s and k-edge-inconnected to s for some $s \in V$. This implies:

Theorem 58.2 (Khuller and Vishkin [28])

Both directed and undirected k-ECSS admit a 2-approximation algorithm.

This method does not work directly for k-CSS, since a graph (digraph) which is k-outconnected from s (and also k-inconnected to s) is usually *not* k-connected. However, many algorithms for k-CSS use an extension of this method, see Sections 58.6 and 58.7.2.

Definition 58.1

*Let $H = (V, E)$ be a (possibly directed) graph. $X \subseteq V$ is an ℓ-*fragment *(in H) if $|\Gamma_H(X)| \leq \ell$ and $V - (X + \Gamma_H(X)) \neq \emptyset$. If H is undirected then $T \subseteq V$ is an ℓ-*fragment transversal *if T intersects every ℓ-fragment. If H is directed then a pair (T^-, T^+) with $T^-, T^+ \subseteq V$ is an ℓ-*fragment transversal *if T^- intersects every ℓ-fragment in H and T^+ intersects every ℓ-fragment in the reverse graph of G.*

When considering k-CSS we will assume that all the graphs are simple. It is well known that in this case, a (directed or undirected) graph $G = (V, E)$ is k-connected if, and only if, either G is a complete graph on $(k + 1)$ nodes or $|V| \geq k + 2$ and G has no $(k - 1)$-fragments, cf. Ref. [38]. An edge e of a graph G is said to be *critical* (with respect to k-connectivity) if G is k-connected but $G - e$ is not. For k-CSS we repeatedly use consequences of the following "Critical Cycle Theorem" due to Mader [39].

Theorem 58.3 (Mader [39])

In a k-connected undirected graph H, any cycle of critical edges has a node v with $d_H(v) = k$.

Corollary 58.1

Let T be a $(k - 1)$-fragment transversal in an undirected graph G_0, and let $E' = \{uv : u \neq v \in T\}$. Then $G_0 + E'$ is k-connected. Moreover, if $|\Gamma(v)| \geq k - 1$ for every $v \in V$, and if $F \subseteq E'$ is an inclusion minimal edge set such that $G_0 + F$ is k-connected, then F is a forest on T and thus $|F| \leq |T| - 1$.

Proof
The first statement follows from Menger's Theorem. For the second statement, note that if F contains a cycle C, then $d_{G_0+F}(v) = d_{G_0}(v) + d_F(v) \geq k + 1$ for every $v \in C$. This contradicts Theorem 58.3. □

We now state the directed counterparts (also due to Mader [40]) of Theorem 58.3 and Corollary 58.1. An even length sequence of directed edges $C = (v_1v_2, v_3v_2, v_3v_4, \ldots, v_{2q-1}v_{2q}, v_1v_{2q})$ in a directed graph H is called an *alternating cycle*; the nodes $v_1, v_3, \ldots, v_{2q-1}$ are *C-out nodes*, and $v_2, v_4, \ldots, v_{2q}$ are *C-in nodes.*

Theorem 58.4 (Mader [40])

In a k-connected directed graph H, any alternating cycle C of critical edges contains a C-in node whose indegree in H is k, or a C-out node whose outdegree in H is k.

Theorem 58.4 implies that if the indegree and the outdegree of every node in H is at least $k - 1$, and if F is an inclusion minimal edge set such that $H + F$ is k-connected, then F contains no alternating cycle. One can associate with every directed graph $J = (V, F)$ an undirected bipartite graph $J' = (V + V', F')$ by adding a copy V' of V and replacing every edge $uv \in F$ by the edge uv'. Mader proved [40] that J has no alternating cycle if, and only if, J' is a forest.

Corollary 58.2

Let (T^-, T^+) be a $(k-1)$-fragment transversal in an directed graph G_0, and let $E' = \{uv : u \in T^-, v \in T^+\}$. Then $G_0 + E'$ is k-connected. Moreover, if the indegree and the outdegree of every node v in G_0 is at least $k - 1$, and if $F \subseteq E'$ is an inclusion minimal edge set such that $G_0 + F$ is k-connected, then F has no alternating cycle and thus $|F| \leq |T^-| + |T^+| - 1$.

58.3 Edge-Covers of Set-Functions and LP-Relaxations

Theorem 58.5 (Generalized Menger's Theorem)

Let u, v be two nodes of a (directed or undirected) graph $G = (V, E)$ and let $S \subseteq V$. Then $\lambda_G^S(u, v) = \min\{|C| : C \subseteq E + S - \{u, v\}, G - C$ has no uv-path$\}$.

This formulation of Menger's Theorem for S-connectivity is easily deduced from its original theorem by standard constructions. Using Theorem 58.5, GSN can be formulated as a *setpair-function edge-cover problem* as follows. $X', X'' \subseteq V$ is a *setpair* (of V) if $X' \cap X'' = \emptyset$; if $V - S \subseteq X' \cup X''$ then (X', X'') is an *S-setpair*. Let us extend the definition of r to setpairs as follows:

$$r(X', X'') = \max\{r(u, v) : u \in X', v \in X''\} \quad \forall\, S\text{-setpair } (X', X''), \quad X', X'' \neq \emptyset \tag{58.2}$$

and $r(X', X'') = 0$ otherwise. Let q be a function defined on setpairs of V by

$$q(X', X'') = \max\{r(X', X'') - (|V| - |X' \cup X''|), 0\} \quad \forall \text{ setpair } (X', X'') \tag{58.3}$$

Then Eq. (58.1) is equivalent to

$$d_G(X', X'') \geq q(X', X'') \quad \forall \text{ setpair } (X', X'') \tag{58.4}$$

It might be the case (e.g., in augmentation problems) that we are already given an initial graph $G_0 = (V, E_0)$ and we seek for a min cost/size set F of links so that Eq. (58.1) holds for $G = G_0 + F$. In this case, let

$$p(X', X'') = \max\{q(X', X'') - d_{G_0}(X', X''), 0\} \quad \forall \text{ setpair } (X', X'') \tag{58.5}$$

Since F is a set of links (i.e., of new edges), then Eq. (58.4) is equivalent to

$$d_F(X', X'') \geq p(X', X'') \quad \forall \text{ setpair } (X', X'') \tag{58.6}$$

Let $p(X', X'')$ be defined by Eq. (58.5). Let $I = \mathcal{E} - E_0$, and associate a variable x_e with every link $e \in I$. Then we get the following LP-relaxation for GSN that has an exponential number of constraints, but it can be solved using the ellipsoid method, and, in many cases, more efficiently by max-flow techniques.

$$\begin{array}{lll} \min & \sum_{e\in I} c_e x_e & \\ \text{s.t.} & \sum_{e\in\delta_I(X',X'')} x_e \geq p(X', X'') & \forall \text{ setpair } (X', X'') \\ & 0 \leq x_e \leq 1 & \forall e \in I \end{array} \tag{58.7}$$

For directed k-OS an appropriate choice of p is $p(X', X'') = k - |V - (X' + X'')|$, if (X', X'') is a setpair with $s \in X'$ and $X'' \neq \emptyset$, and $p(X', X'') = 0$ otherwise. Frank and Tardos [36] proved that linear program (58.7) always has an optimal solution which is integral. This can be used to show that:

Lemma 58.1

Let G_0 be an ℓ-outconnected from s subgraph of cost zero of a directed k-outconnected from s graph $\mathcal{G}$, $\ell < k$. Then the minimum cost of an $(\ell+1)$-outconnected spanning subgraph of $\mathcal{G}$ is at most $\frac{1}{k-\ell}$ times the minimum cost of a k-outconnected from s spanning subgraph of $\mathcal{G}$.

We now consider edge-connectivity problems. In this case $S = \emptyset$, and thus (X', X'') is an S-setpair if, and only if, $X'' = V - X'$; in particular, $p(X', X'') > 0$ implies $X'' = V - X'$. Thus p can be considered as a *set-function* on subsets of V, where its value on X is $p(X, V - X)$. Similarly, r can be considered as a set function where its value on X is $r(X, V - X)$. Then Eq. (58.6) and linear program (58.7) can be rewritten as

$$d_F(X) \geq p(X) \equiv \max\{r(X) - d_{G_0}(X), 0\} \quad \forall\, X \subseteq V \tag{58.8}$$

$$\begin{array}{lll} \min & \sum_{e\in I} c_e x_e & \\ \text{s.t.} & \sum_{e\in\delta_I(X)} x_e \geq p(X) & \forall\, X \subseteq V \\ & 0 \leq x_e \leq 1 & \forall e \in I \end{array} \tag{58.9}$$

Definition 58.2

A set function p is skew-supermodular *(or* weakly supermodular*) if for every $X, Y \subseteq V$ at least one of the following holds:*

$$p(X) + p(Y) \leq p(X \cap Y) + p(X \cup Y) \tag{58.10}$$

$$p(X) + p(Y) \leq p(X - Y) + p(Y - X) \tag{58.11}$$

If Eq. (58.10) always holds whenever $X \cap Y \neq \emptyset$ and $X \cup Y \neq V$, then p is crossing supermodular.

Lemma 58.2

Let $G_0 = (V, E_0)$ be a (possibly directed) graph, let r be a requirement function on $V \times V$, and let $p(X) = r(X) - d_{G_0}(X)$ for all $X \subseteq V$.

(i) For undirected G_0 p is skew-supermodular.
(ii) For both directed an undirected G_0, if $r(X) = k$ for all $X \subseteq V$ then p is crossing supermodular.

The following concepts are used in Sections 58.5.2 and 58.7.1. Let x belong to a polyhedron $P \subseteq R^m$ defined by a system $\mathcal{I}$ of linear inequalities; an inequality in $\mathcal{I}$ is *tight* (for x) if it holds as equality for x. $x \in P$ is a *basic solution* for the system defining P if there exists a set of m tight inequalities from the system defining P such that x is the unique solution for the corresponding equation system; that is, the corresponding m tight equations are linearly independent. It is well known that if the problem $\min\{cx : x \in P\}$ has an optimal solution, then it has an optimal solution which is basic. Let x be an arbitrary basic solution for linear program (58.9), and consider the corresponding m tight linearly independent equations. We will be particularly interested in "fractional" solutions with $0 < x_e < 1$ for all $e \in I$; in this case, every tight equation is of the form $\sum_{e \in I} x_e = p(X)$, and then we say that X is *tight* (for x). Let us say that a family of tight sets on V is x*-defining* if x is the unique solution for $\{\sum_{e \in \delta_I(X)} x_e = p(X), \forall X \in \mathcal{F}\}$. A family $\mathcal{F}$ of sets is *laminar* if for every $A, B \in \mathcal{F}$, either $A \cap B = \emptyset$, or $A \subseteq B$, or $B \subseteq A$. Part (i) of the following statement is from Ref. [25] and part (ii) from Ref. [41].

Theorem 58.6 **(Jain [25], Melkonian and Tardos [41])**

Let x be a basic feasible solution for linear program (58.9) *and assume that* $0 < x_e < 1$ *for all* $e \in I$.

(i) If I is undirected and p is skew-supermodular then there exists an x-defining family which is laminar.
(ii) If I is directed and p is crossing supermodular, then there exists an x-defining family $\mathcal{F}$ and $\mathcal{O} \subseteq \mathcal{F}$ such that if $\mathcal{I} = \{V - X : X \in \mathcal{F} - \mathcal{O}\}$ then the family $\mathcal{I} + \mathcal{O}$ is laminar.

58.4 Connectivity Augmentation Problems ({0, 1}-Costs)

58.4.1 An $O(\log n)$-Approximation Algorithm for Arbitrary Requirements

Here we present the result of Ref. [22] for the following problem (for surveys of the cases that are in P see [8,17]):

Connectivity Augmentation

Instance: A directed/undirected graph $G_0 = (V, E_0)$, $S \subseteq V$, and a requirement function $r(u, v)$ on $V \times V$.

Objective: Find a minimum-size set F of links so that $\lambda^S_{G_0+F}(u, v) \geq r(u, v)$ for all $(u, v) \in V \times V$.

Theorem 58.7 **(Kortsarz and Nutov [22])**

Connectivity augmentation (CA) admits an $O(\log n)$*-approximation algorithm except the case* $S = V$ *for which there exists an* $O(r_{\max} \log n)$*-approximation algorithm.*

For $S \neq V$ the approximation ratio in Theorem 58.7 is tight since the problem has an $\Omega(\ln n)$-approximation threshold [34] (for directed graphs even for rooted $\{0, 1\}$-requirements, see Theorem 58.21). For $S = V$, the approximation ratio in Theorem 58.7 is tight for small requirements, but may seem weak if $r_{\max}$ is large. However, a much better approximation algorithm might not exist; for $S = V$ CA with $r(u, v) \in \{0, k\}$ $(k = \Theta(n))$ cannot be approximated within $2^{\log^{1-\varepsilon} n}$ for any $\varepsilon > 0$ unless NP $\subseteq$ DTIME$(n^{\text{polylog}(n)})$ [7]. We note that Theorem 58.7 is also unlikely to be extended to $\{0, 1, \infty\}$-costs case, see Theorem 58.23.

The proof of Theorem 58.7 follows. We prove Theorem 58.7 for the directed case, and the statement for the undirected CA follows from Proposition 58.1. Let F' be an arbitrary solution for an instance G_0, S, r of directed CA. Subdivide every edge in F' by a new node, and then identify all these new nodes into a node s. The obtained graph satisfies the requirements between nodes in V, and the number of links incident to s is $2|F'|$. If $V - S \neq \emptyset$, then by identifying s with some node $v \in V - S$ we get that the links added form a feasible solution for G_0, S, r. This implies:

Corollary 58.3

For any solution F' for directed CA *with $S \neq V$ and any $s \in V - S$, there exists a solution F with $|F| \leq 2|F'|$ such that all the links in F are incident to s.*

If $S = V$, we make $r_{\max}$ copies $s_1, \ldots, s_{r_{\max}}$ of s and of the links incident to s, choose arbitrary $r_{\max}$ nodes $\{v_1, \ldots, v_{r_{\max}}\}$, and identify every s_i with v_i. Again, it is easy to see that the new links added form a feasible solution to the CA instance, and the number of links added is $2|F'|r_{\max}$.

Given an instance G_0, S, r for directed CA, let $H_0 = G_0 + s$ (note that $s \notin S$). We say that a set F of links incident to s is a feasible solution for H_0 if $H_0 + F$ satisfies the S-connectivity requirements defined by r. *The H_0-problem* is to find a feasible solution for H_0 of minimum size. We show an $O(\log n)$-approximation algorithm for the H_0-problem. This is done by approximating the following two problems.

Let H_0^+ be obtained from H_0 by adding $r_{\max}$ edges from s to every node in V, and H_0^- is obtained by adding $r_{\max}$ edges from every node in V to s. A set F^+ (F^-) of links entering (leaving) s is a feasible solution for H_0^+ (for H_0^-) if $H_0^+ + F^+$ (if $H_0^- + F^-$) satisfies the S-connectivity requirements defined by r. The H_0^+-problem is to find a feasible solution for H_0 of minimum size opt^+, and the H_0^- problem is defined similarly.

Lemma 58.3

Let F^+ and F^- be a feasible solution for H_0^+ and for H_0^-, respectively. Then $F^+ + F^-$ is a feasible solution for the H_0 problem.

Lemma 58.4

The H_0^+-problem (and the H_0^--problem) admits an $O(\log n)$-approximation algorithm.

The algorithm for directed CA with $S \neq V$ is as follows:

(1) Using the algorithm from Lemma 58.4 find a solutions F^+ for the H_0^+-problem and F^- for the H_0^--problem, so that $|F^+| = O(\log n)\mathsf{opt}^+$ and $|F^-| = O(\log n)\mathsf{opt}^-$.

(2) Let $F = F^+ + F^-$, and let $H = H_0 + F$.
Obtain a graph G from H by identifying s with an arbitrary node in $V - S$.

The algorithm computes a feasible solution, by Corollary 58.3 and Lemma 58.3. The approximation ratio is $O(\log n)$, by Lemma 58.4. To complete the proof of Theorem 58.7 it remains to prove Lemmas 58.3 and 58.4. We need the following statement that stems from Menger's Theorem.

Fact 58.1

$\lambda_H^S(u, v) \geq r(u, v)$ if, and only if, $|Q| + d_H(X, Y) \geq r(u, v)$ for any partition X, Q, Y of the node set of H with $u \in X$, $v \in Y$, and $Q \subseteq S$.

Proof of Lemma 58.3

Let $H = H_0 + F$. Suppose to the contrary there are $u, v \in V$ so that $\lambda_H^S(u, v) < r(u, v)$. Then by Fact 58.1 there exists a partition X, Q, Y of $V + s$ with $u \in X$, $v \in Y$, and $Q \subseteq S$ such that $|C| < r(u, v)$ for $C = Q + \delta_H(X, Y)$. Note that $s \notin C$, so $s \in X$ or $s \in Y$. If $s \in X$ then $\delta_{H^-}(X, Y) = \delta_H(X, Y)$, where $H^- = H_0^- + F^-$, so $H^- - C$ has no uv-path. Since $|C| < r(u, v)$, we get that $\lambda_{H^-}^S(u, v) < r(u, v)$, contradicting that F^- is a feasible solution for H_0^-. The proof for the case $s \in Y$ is similar. □

In the rest of this section we prove Lemma 58.4. We use a result due to Wolsey [42] about the performance of the greedy algorithm for a certain type of covering problems. A *Covering Problem* is defined as follows:

Instance: An integer nondecreasing function p given by an evaluation oracle on subsets of a groundset I.
Objective: Find $F \subseteq I$ of minimum size so that $p(F) = p(I)$.

The *Greedy Algorithm* starts with $F = \emptyset$ and adds elements to the solution one after the other using the following simple greedy rule. As long as $p(F) < p(I)$ it adds to F an element $e \in I$ that has maximum $p(F+e) - p(F)$; if this step can be performed in polynomial time, then the algorithm can be implemented in polynomial time. Let $\Delta_p = \max_{e \in I}(p(e) - p(\emptyset))$, and for an integer k let $H(k)$ be the kth Harmonic number.

Theorem 58.8 (Wolsey [42])

Suppose that for an instance of a covering problem

$$\sum_{e \in F_2} (p(F_1 + e) - p(F_1)) \geq p(F_1 + F_2) - p(F_1) \quad \forall F_1, F_2 \subseteq I, F_1 \cap F_2 = \emptyset \tag{58.12}$$

Then the Greedy Algorithm produces a solution of size at most $H(\Delta_p)$ times the optimal.

We formulate the H_0^+-problem as a covering problem and using Theorem 58.8 show that it admits an $O(\log n)$-approximation algorithm. The set I is obtained by taking $r_{\max}$ links from v to s for every $v \in V$. We also need to define a function p on the subsets of I. For $(u, v) \subseteq V \times V$ and $F^+ \subseteq I$, let $q(F^+, (u, v)) = \max\{r(u, v) - \lambda^S_{H_0^+ + F^+}(u, v), 0\}$ be the deficiency of (u, v) in $H_0^+ + F^+$. Let

$$q(F^+) = \sum_{(u,v)\in V\times V} q(F^+, (u, v))$$

be the total deficiency of $H_0^+ + F^+$. Then $p(F^+) = q(\emptyset) - q(F^+)$. In other words, $p(F^+)$ is the decrease in the total deficiency as a result of adding F^+ to H_0^+; in the corresponding covering problem, the goal is to find a minimum size $F^+ \subseteq I$ so that $p(F^+) = p(I)$ (that is, $q(F^+) = 0$). Clearly p is increasing. The Greedy Algorithm can be implemented in polynomial time, as $p(F^+)$ can be computed in polynomial time for any link set F^+. Clearly, $\Delta_p \leq n^2$. We prove that Eq. (58.12) holds for p, and thus Theorem 58.8 implies that the Greedy Algorithm produces a solution of size at most $H(\Delta_p) \cdot \mathsf{opt}^+ \leq H(n^2) \cdot \mathsf{opt}^+ = O(\log n) \cdot \mathsf{opt}^+$.

Let $F_1, F_2 \subseteq I$ be disjoint link sets. To simplify the notation, denote $J = H_0^+ + F_1$, $F = F_2$, and denote by $\Delta(F, (u, v))$ the decrease in the deficiency of (u, v) as a result of adding F to J. Namely, $\Delta(F, (u, v))$ is obtained by subtracting the deficiency of (u, v) in $J + F$ from the deficiency of (u, v) in J. Then Eq. (58.12) is equivalent to

$$\sum_{e\in F} \sum_{(u,v)\in V\times V} \Delta(e, (u, v)) \geq \sum_{(u,v)\in V\times V} \Delta(F, (u, v))$$

Consequently, it would be sufficient to show that

$$\sum_{e\in F} \Delta(e, (u, v)) \geq \Delta(F, (u, v)) \quad \forall (u, v) \in V \times V \tag{58.13}$$

Let $u, v \in V$. If $\lambda^S_J(u, v) \geq r(u, v)$, then Eq. (58.13) is valid, since both its sides are zero. Note that $\lambda_{J+F}(u, v) - \lambda_J(u, v) \geq \Delta(F, (u, v))$, while $\Delta(e, (u, v)) = \lambda^S_{J+e}(u, v) - \lambda^S_J(u, v)$ if $\lambda^S_J(u, v) \leq r(u, v) - 1$. Thus if $\lambda^S_J(u, v) \leq r(u, v) - 1$, it would be sufficient to prove that for any link set F entering s:

$$\sum_{e\in F} \left(\lambda^S_{J+e}(u, v) - \lambda^S_J(u, v)\right) \geq \lambda_{J+F}(u, v) - \lambda_J(u, v) \quad \forall (u, v) \in V \times V$$

Let us say that $X \subseteq V$ is (u, v)-*tight* (in J) if there exists a partition X, Q, Y of V with $u \in X, v \in Y$, and $Q \subseteq S$ such that $|Q| + d_J(X, Y) = \lambda^S_J(u, v)$. It is well known and easy to show that:

Fact 58.2

The intersection and union of two (u, v)*-tight sets are also* (u, v)*-tight.*

For $u \in V$ let X_u be the unique minimal (u, v)-tight set in J. By Fact 58.1 and the definition of J, $\lambda^S_{J+e}(u, v) - \lambda^S_J(u, v) = 1$ if e connects X_u with s. Let $t = \lambda^S_{J+F}(u, v) - \lambda^S_J(u, v)$. Then at least t links in F must connect X_u with s. Thus, each one of these t links contributes 1 to $\sum_{e\in F} \left(\lambda^S_{J+e}(u, v) - \lambda^S_J(u, v)\right)$. This finishes the proof of Lemma 58.4, and the proof of Theorem 58.7 is complete.

58.4.2 Augmenting a k-Connected Graph to Be $(k + 1)$-Connected

Recall that a simple graph is k-connected if there are k pairwise internally disjoint paths between every pair of its nodes. We describe a version of Jordán's algorithm [38,43] from Ref. [44] for the following problem:

Instance: A k-(node) connected graph G.
Objective: Find a smallest set F of links so that the graph $G + F$ is $(k + 1)$-connected.

The complexity status of this problem is a major open question in graph connectivity. Recall that a similar problem for digraphs is solvable in polynomial time [10], and this implies a 2-approximation algorithm for undirected graphs. Jordán's algorithm computes an augmenting edge set with at most $\lceil (k-1)/2 \rceil$ edges over (a lower bound of) the optimum. The following property of k-fragments (cf. Ref. [38]) is used.

Lemma 58.5

Let X, Y be two intersecting k-fragments in a k-connected graph G. If $V - (X + Y + \Gamma(X+Y)) \neq \emptyset$ or if $|V - (X+Y)| \geq k$, then $X \cap Y$ is also a k-fragment.

It follows from Menger's Theorem that $G + F$ is $(k+1)$-connected if, and only if, F has a link between X and $V - (X + \Gamma(X))$ for every k-fragment X of G. Henceforth, let T be an arbitrary inclusion minimal k-fragment transversal in G; such T can be computed in polynomial time using max-flow techniques.

Lemma 58.6

$\mathsf{opt} \geq \lceil |T|/2 \rceil$. *Furthermore, if $|T| \geq k+2$ then the minimal k-fragments are disjoint.*

Proof
Let $\mathcal{F}(G)$ denote the family of inclusion minimal k-fragments of G. Clearly $|\mathcal{F}(G)| \geq |T|$. We will prove that $\mathsf{opt} \geq \lceil |\mathcal{F}(G)|/2 \rceil$. For that, it would be enough to show that $|\mathcal{F}(H+e)| \geq |\mathcal{F}(H)| - 2$ for any k-connected graph H and link e. If not, then there is a link $e = uv$ and $X, Y \in \mathcal{F}(H)$ such that $u \in X \cap Y$ and $v \in V - (X + Y + \Gamma(X+Y))$. By Lemma 58.5 $X \cap Y$ is also a k-fragment of H, contradicting the minimality of X, Y. Now suppose that $|T| \geq k+2$. The minimality of T implies that for every $u \in T$ there exists $X_u \in \mathcal{F}(G)$ with $|X_u \cap T| = \{u\}$. If the sets $\{X_u : u \in T\}$ are pairwise disjoint, the statement is obvious. Suppose therefore that there are X_u and X_v that intersect. If $|T| \geq k+2$, then $|V - (X_u \cup X_v)| \geq |T| - 2 \geq k$, and thus by Lemma 58.5 their intersection is also a k-fragment, contradicting the minimality of X_u, X_v. □

Another lower bound on opt is as follows. For $C \subseteq V$ the *C-components* are the connected components of $G - C$ and let $b(C)$ denote the number of C-components; C is a *k-separator* of G if $|C| = k$ and $b(C) \geq 2$. A k-separator C is a *k-shredder* if $b(C) \geq 3$. All k-shredders separating two given nodes u, v can be found using one max-flow computation, as follows. First, compute a set Π of k internally disjoint uv-paths, and set P to be the the union of their nodes. Second, for every connected component X of $G - (P - \{u, v\})$ check whether $\Gamma(X)$ is a k-shredder. The algorithm is correct since if C is a k-shredder so that u, v belong to distinct C-components, then every C-component X with $X \cap \{u, v\} = \emptyset$ is a connected component of $G - (P - \{u, v\})$; this is so since $C \subseteq P - \{u, v\}$ and $X \cap P = \emptyset$. Indeed, any uv-path that contains a node from X goes through C at least twice, and thus contains at least two nodes from C, but since $|C| = |\Pi|$ and the paths in P are internally disjoint this is not possible for a path from Π.

Let $b(G) = \max\{b(C) : C \subseteq V, |C| = k\}$. If $G + F$ is $(k+1)$-connected then $|F| \geq b(G) - 1$, since for any k-separator C, F must induce a connected graph on the C-components. Combining with Lemma 58.6 gives

$$\mathsf{opt} \geq \max\{\lceil |T|/2 \rceil, b(G) - 1\} \tag{58.14}$$

Theorem 58.9 (Jordán [38,43])

There exists a polynomial-time algorithm that given a k-connected graph $G = (V, E)$ finds an augmenting edge set F with $|F| \leq \mathsf{opt} + \lceil (k-1)/2 \rceil$ such that $G + F$ is $(k+1)$-connected. Moreover, for any minimal k-fragment transversal T of G the following holds: $|F| = \max\{\lceil |T|/2 \rceil, b(G) - 1\} = \mathsf{opt}$ if $b(G) \geq k+1$, and $|F| \leq \lceil |T|/2 \rceil + \lceil (k-1)/2 \rceil$ if $b(G) \leq k$ and $|V| \geq 2k+1$.

A link uv with $u, v \in T$ is *(G, T)-legal* if $T - \{u, v\}$ is a k-fragment transversal of $G + uv$. Intuitively, whenever $|T|/2 > b(G) - 1$ the idea is to add a single link and reduce the size of the traversal by 2. One can find a (G, T)-legal pair or determine that such does not exist in polynomial time using max-flow techniques. The algorithm relies on the following key theorem (part (i) is from Ref. [44] and part (ii) is from Ref. [38]).

Theorem 58.10 (Liberman and Nutov [44], Jordán [38])

Let T be a minimal k-fragment transversal of a k-connected graph $G = (V, E)$. Then:

(i) If C is a k-shredder of G with $b(C) = b(G) \geq k+1$ and if X is a C-component with $|T \cap X| \geq b(C)$ then there exists a (G, T)-legal link $e = uv$ with $u, v \in X \cap T$.
(ii) If $|V| \geq 2k+1$ and $|T| \geq k+3$ then either $b(G) = |T|$, or there exists a (G, T)-legal link.

The case $b(G) \geq k+1$: Let C be a k-shredder with $b(C) = b(G) \geq k+1$. Then $X \cap C = \emptyset$ for any minimal tight set X, as otherwise it can be shown that X has a neighbor in every C-component, which gives a contradiction $k = |\Gamma(X)| \geq b(C) \geq k+1$. In particular, $T \cap C = \emptyset$. Thus every minimal k-fragments is contained in some C-component, and (by Lemma 58.5) the minimal k-fragments are pairwise disjoint. Let us say that a link set F on T is a *(C, T)-connecting cover* if the following three conditions hold: (a) $d_F(v) \geq 1$ for every $v \in T$; (b) every edge in F connects distinct C-components; (c) F induces a connected graph on the C-components. Let $\max(C, T) = \max\{|T \cap X| : X \text{ is a } C\text{-component}\}$. In Ref. [44] it is proved:

Lemma 58.7

If $b(C) \geq k+1$ then any (C, T)-connecting cover is a feasible solution. Furthermore, an optimal (C, T)-connecting cover of size $\max\{\lceil |T|/2 \rceil, \max(C, T), b(C) - 1\}$ *can be found in polynomial time.*

The following algorithm finds an (optimal) augmenting edge set F of size $\max\{\lceil |T|/2 \rceil, b(G) - 1\}$.

```
Phase 1: While there exists a $C$-component $X$ with $|T \cap X| \geq b(C)$ do:
             Find a $(G, T)$-legal link $uv$ with $u, v \in X$ and set $G \leftarrow G + uv$, $T \leftarrow T - \{u, v\}$;
         End While
Phase 2: Add to $G$ a minimum size $(C, T)$-connecting cover.
```

The condition in the loop of phase 1 ensures that an appropriate (G, T)-legal link exists, by Theorem 58.10(i). Consequently, the algorithm is correct, by Lemma 58.7. Let F_1 and F_2 be the link sets added at phases 1 and 2, respectively. We show that $|F_1| + |F_2| = \max\{\lceil |T|/2 \rceil, b(C) - 1\} = \mathsf{opt}$. If $F_1 = \emptyset$ then $\max(C, T) \leq b(C) - 1$, and thus by Lemma 58.7 $|F| = |F_2| = \max\{\lceil |T|/2 \rceil, b(C) - 1\}$. Assume therefore that $F_1 \neq \emptyset$. Let T_2 be the set of nodes in T when phase 2 starts. Clearly $|T_2| = |T| - 2|F_1|$. We claim that $|F_2| = \lceil |T_2|/2 \rceil$ and thus $|F_1| + |F_2| = |F_1| + \lceil (|T| - 2|F_1|)/2 \rceil = \lceil |T|/2 \rceil = \mathsf{opt}$. To see that $|F_2| = \lceil |T_2|/2 \rceil$, note that there is a C-component X with $|X \cap T_2| \geq b(C) - 2$, while $|Y \cap T_2| \geq 1$ for any other C-component Y, so $|T_2| \geq (b(C) - 2) + (b(C) - 1) = 2b(C) - 3$. Consequently, $|F_2| = \max\{\lceil |T_2|/2 \rceil, b(C) - 1\} = \lceil |T_2|/2 \rceil$.

The case $b(G) \leq k$: The following statement from Ref. [43] is used when $|V| \leq 2k$.

Lemma 58.8

Let G be a k-connected graph with $|V| \leq 2k$, and let $F_1 = \{u_1v_1, \ldots, u_jv_j\}$ be a sequence of links such that u_iv_i is (G_i, T_i)-legal where for $i = 1, \ldots, j$: $G_1 = G$, $T_1 = T$, $G_{i+1} = G_i + u_iv_i$, and $T_{i+1} = T_i - \{u_i, v_i\}$. If $|T_{j+1}| \geq k+3$ and if no (G_{j+1}, T_{j+1})-legal link exists, then one can find in polynomial time a link set F_2 so that $G + F_1 + F_2$ is $(k+1)$-connected and $|F_1| + |F_2| \leq \mathsf{opt} + \lceil (k-1)/2 \rceil$.

Here is a description of the algorithm for the case $b(G) \leq k$.

```
Phase 1: While $|T| \geq k+3$ and there exists a $(G, T)$-legal link $uv$ do:
             $G \leftarrow G + uv$, $F \leftarrow F + uv$, $T \leftarrow T - \{u, v\}$.
         End While
Phase 2: If $|T| \leq k+2$ add to $G$ a forest on $T$ as in Corollary 58.1;
         Else ($|V| \leq 2k$) add to $G$ an augmenting edge set as in Lemma 58.8.
```

We now finish the proof of Theorem 58.9 for the case $b(G) \leq k$. Let F_1 and F_2 be the link sets added phases 1 and 2, respectively. Let T_2 be the set of nodes in T when phase 2 starts. The case $|T_2| = 0$ is obvious, while $|T_2| = 1$ is not possible. Assume therefore that $|T_2| \geq 2$. If $|T_2| \leq k+2$ then:

$$|F_1| + |F_2| = (|T| - |T_2|)/2 + (|T_2| - 1) = \lceil |T|/2 \rceil + \lceil (|T_2| - 1)/2 \rceil - 1 \leq \lceil |T|/2 \rceil + \lceil (k-1)/2 \rceil$$

If $|T_2| \geq k+3$, then $|V| \leq 2k$, by Theorem 58.10 (ii). The correctness of this case follows from Lemma 58.8.

58.5 Min-Size k-Connected Spanning Subgraphs ($\{1, \infty\}$-Costs)

58.5.1 Algorithm Based on Edge-Covers

Here we consider simple graphs only and survey the results from Ref. [24] (see Ref. [23] for the case of multigraphs).

Theorem 58.11 **(Cheriyan and Thurimella [24])**

Both directed and undirected min-size k-CSS admit a $(1 + 1/k)$-approximation algorithm. The undirected min-size k-ECSS admits a $(1 + 2/(k+1))$-approximation algorithm.

The proof of Theorem 58.11 relies on two lower bounds on the optimum. The first lower bound is as follows. Let $G = (V, E)$ be a graph and let $n = |V|$. Note that if $G = (V, E)$ is k-edge-connected then $d_G(v) \geq k$ for every $v \in V$; thus $|E| \geq kn/2$ if G is undirected and $|E| \geq kn$ if G is directed. The same is true if G is k-connected, since then G is also k-edge connected. This implies the following lower bound on opt for min-size k-CSS and k-ECSS: $\mathsf{opt} \geq kn/2$ for undirected graphs and $\mathsf{opt} \geq kn$ for directed graphs.

The above lower bound can be used to get a 2-approximation algorithm for both directed and undirected k-CSS and k-ECSS. Let G be a *minimally k-connected graph*, that is, G is k-connected, but $G - e$ is not k-connected for any edge e of G. If G is undirected then G has at most kn edges, by Corollary 58.1. Similarly, if G is directed then G has at most $2kn$ edges, by Corollary 58.2. These bounds extend to edge-connectivity as well. Thus by simply taking a minimally k-connected (k-edge-connected) graph we obtain a 2-approximation algorithm, for the directed and undirected min-size k-CSS (k-ECSS).

One can improve on this using the following idea. For undirected graphs an edge set E_0 on V is an ℓ-edge cover (of V) if $d_{E_0}(v) \geq \ell$ for every $v \in V$; for directed graphs we require that both the indegree and the outdegree of every node is at least ℓ. For both directed and undirected graphs a minimum size ℓ-edge cover can be computed in polynomial time, since it is a complementary problem of the b-matching problem, cf., Ref. [1]. The algorithm for (both directed and undirected) min-size k-CSS is as follows.

Phase 1: Find a minimum size $(k-1)$-edge cover $E_0 \subseteq \mathcal{E}$.
Phase 2: Find an inclusion minimal edge set $F \subseteq \mathcal{E} - E_0$ so that $G_0 + F$ is k-connected.

Clearly $|E_0| \leq \mathsf{opt}$. For undirected graphs $|F| \leq n - 1 \leq 2\mathsf{opt}/k$, while for directed graphs $|F| \leq 2n - 1 \leq 2\mathsf{opt}/k$. Thus $|F| \leq 2\mathsf{opt}/k$ for both directed and undirected graphs. Consequently, the size of the subgraph computed by the algorithm is bounded by $|E_0| + |F| \leq \mathsf{opt} + 2\mathsf{opt}/k = \mathsf{opt}(1 + 2/k)$.

The following key theorem from Ref. [24] enables to improve the approximation ratio from $1 + 2/k$ to $1 + 1/k$.

Theorem 58.12 **(Cheriyan and Thurimella [24])**

Let $G = (V, E)$ be an undirected graph with minimum degree $\geq k$, and let $E_0 \subseteq \mathcal{E}$ be a minimum size $(k-1)$-edge edge cover. If G is k-connected or bipartite, then $|E| \geq |E_0| + \lfloor |V|/2 \rfloor$.

Using the improved lower bound provided by Theorem 58.12 we get for undirected graphs:

$$|E_0| + |F| \leq (\mathsf{opt} - \lfloor n/2 \rfloor) + (n-1) \leq \mathsf{opt} + n/2 \leq (1 + 1/k)\mathsf{opt}$$

For directed graphs, a similar analysis on the associated bipartite graph gives

$$|E_0| + |F| \leq (\mathsf{opt} - \lfloor 2n/2 \rfloor) + (2n-1) \leq \mathsf{opt} + (n-1) \leq (1 + 1/k)\mathsf{opt}$$

We now turn to the undirected min-size k-ECSS. For this case the algorithm is almost the same, but at phase 1 we find a minimum size k-edge-cover $E_0 \subseteq \mathcal{E}$ (instead of a $(k-1)$-edge cover). The proof of the approximation ratio is based on the following statement from Ref. [24].

Theorem 58.13 (Cheriyan and Thurimella [24])

Let G_0 be an undirected graph of minimal degree $\geq k$, and let F be an inclusion minimal edge set so that $G_0 + F$ is k-connected. Then $|F| \leq kn/(k+1)$.

The approximation ratio $(1 + 2/(k+1))$ follows, since $|F| \leq 2\mathsf{opt}/(k+1)$ and thus $|E_0| + |F| \leq \mathsf{opt} + 2\mathsf{opt}/(k+1) = (1 + 2/(k+1))\mathsf{opt}$. This completes the proof of Theorem 58.11.

58.5.2 LP-Rounding Algorithm for Directed Min-Size k-ECSS

Theorem 58.14 (Gabow, Goemans, Tardos, and Williamson [23])

Directed min-size k-ECSS admits a $(1 + 2/k)$-approximation algorithm.

The proof of Theorem 58.14 (due to Ref. [23]) follows. The algorithm computes a basic optimal solution y to Eq. (58.9) with $p(X) = k$ for all $\emptyset \neq X \subset V$, and outputs $G = (V, E)$ where $E = \{e : y_e > 0\}$. Clearly, the derived solution is feasible. Let us partition E into $F = \{e : 0 < y_e < 1\}$ and $E_0 = \{e : y_e = 1\}$. Let x be the restriction of y to F. Let $x(F) = \sum_{e \in F} x_e$, and $\mathsf{opt}^* = |E_0| + x(F)$ be the optimal (fractional) value of linear program (58.9). Clearly, $\mathsf{opt}^* \geq kn$ and thus the approximation ratio ρ (and the integrality gap) is bounded by:

$$\rho = \frac{|E_0| + |F|}{|E_0| + x(F)} = 1 + \frac{F - x(F)}{E_0 + x(F)} = 1 + \frac{|F| - x(F)}{\mathsf{opt}^*} \leq 1 + \frac{|F| - x(F)}{kn} \tag{58.15}$$

Let $G_0 = (V, E_0)$. Then x is an optimal basic solution to linear program (58.9) with p defined by Eq. (58.8). By Lemma 58.2 p is crossing supermodular. By Theorem 58.6(ii) there exists an x-defining family $\mathcal{F}$ and $\mathcal{O} \subseteq \mathcal{F}$ such that if $\mathcal{I} = \{V - X : X \in \mathcal{F} - \mathcal{O}\}$ then the family $\mathcal{I} + \mathcal{O}$ is laminar; each one of the families $\mathcal{I}$, $\mathcal{O}$ consists from distinct sets, but it might be that the same set belongs both to $\mathcal{I}$ and to $\mathcal{O}$. This implies a $(1 + 4/k)$-approximation ratio: we claim that $|F| = |\mathcal{I}| + |\mathcal{O}| \leq 4n - 2$, and thus by Eq. (58.15) the approximation ratio is bounded by $1 + (4n-2)/kn \leq 1 + 4/k$. Indeed, $|F| = |\mathcal{I}| + |\mathcal{O}|$ since $|\mathcal{F}| = |\mathcal{I}| + |\mathcal{O}|$ for the family $\mathcal{F}$ that is x-defining (if a set of equations has a unique solution then the number of equations equals the number of variables). It is well known that a laminar family (of distinct sets) on $|V|$ has at size at most $2|V| - 1$; thus $|\mathcal{I}|, |\mathcal{O}| \leq 2n - 1$.

We describe the improved analysis of Ref. [23]. By Eq. (58.15), Theorem 58.14 will be proved if we show that

$$|F| \leq 2n + x(F) \tag{58.16}$$

Let $\mathcal{L}$ be a laminar family. We say that X *owns* v *in* $\mathcal{L}$ if X is the inclusion minimal set in $\mathcal{L}$ that contains v. Define $\phi(X)$ to be the sum of x_e over all $e = uv \in F$ so that X owns u and v in $\mathcal{F} = \mathcal{I} + \mathcal{O}$.

Lemma 58.9

Let $X \in \mathcal{I}$ and suppose that X does not own any node in $\mathcal{I}$. Then $\phi(X)$ is a positive integer. Furthermore, if $X \in \mathcal{I} \cap \mathcal{O}$ then X owns some node in $\mathcal{O}$.

Now, consider the contribution of every set X to both sides of Eq. (58.16). Sets X that own a node v either in $\mathcal{I}$ or in $\mathcal{O}$ contribute at most $2n$ to $|\mathcal{I}| + |\mathcal{O}|$ and this accounts for the $2n$ term in the r.h.s. If $X \in \mathcal{I}$ and does not own a vertex in $\mathcal{I}$ nor in $\mathcal{O}$ then $X \in \mathcal{I} \backslash \mathcal{O}$ and $\phi(X) \geq 1$, by Lemma 58.9. Such X contributes 1 to the l.h.s of Eq. (58.16) and at least 1 to its r.h.s.

58.6 Algorithms for *k*-CSS with Metric Costs

The first constant approximation ratio for undirected metric k-CSS is due to Khuller and Raghavachari [45]. We present a slightly improved version, as well as an algorithm for directed graphs from Ref. [12].

Theorem 58.15 (Kortsarz and Nutov [12])

Undirected metric k-CSS admits a $(2+\frac{k-1}{n})$-approximation algorithm. Directed metric k-CSS admits a $(2+\frac{k}{n})$-approximation algorithm.

We use the following lemma from Ref. [9], which is valid for both directed and undirected graphs.

Lemma 58.10

Let X be an ℓ-fragment of a k-inconnected to s graph H with $s \notin X$ and let $T = \{v \in V : s \in \Gamma_H(v)\}$. If $s \in \Gamma_H(X)$ then $|X \cap T| \geq k - \ell + 1$, and if $s \notin \Gamma_H(X)$ then $\ell \geq k$. Thus T is a $(k-1)$-fragment transversal of $H - s$.

Proof

Let $v \in X$, and consider a set of k internally disjoint vs-paths in H. Let $T' = \{v_1, \ldots, v_k\} \subseteq T$ be the nodes of these paths preceding s. If $s \in \Gamma_H(X)$, then at most $\ell - 1$ nodes from T' may not belong to X; this implies $|T \cap X| \geq |T' \cap X| \geq k - (\ell - 1)$. Clearly, if $s \notin T$ and $\ell < k$ there cannot be k internally disjoint vs-paths, by Menger's Theorem. The last statement follows from the simple observation that if X is a $(k-1)$-fragment of $H - s$ but not of H, then X is a k-fragment of H with $s \in \Gamma_H(X)$. □

58.6.1 Undirected Graphs

A tree J on ℓ nodes with a designated *center* v is a *$v - \ell$-star* if every node of J distinct from v is a leaf. Among all subdigraphs of $\mathcal{G}$ which are $v - \ell$-stars, let $J_\ell(v)$ be a cheapest one; clearly, $J_\ell(v)$ can be computed in polynomial time. The algorithm for undirected graphs is as follows:

(1) Find a node v_0 for which $c(J_{k+1}(v_0))$ is minimal.
Let $\{v_1, \ldots, v_k\}$ be the leaves of $J_{k+1} = J_{k+1}(v_0)$, where $c(v_0v_i) \leq c(v_0v_{i+1})$, $i = 1, \ldots, k-1$.

(2) Set $T = \{v_0, \ldots, v_{k-1}\}$ (note that $v_k \notin T$) and add a node s to $\mathcal{G}$ and edges $\{vs : v \in T\}$ of the cost 0, obtaining a graph $\mathcal{G}_s$; compute a k-outconnected from s subgraph H_s of $\mathcal{G}_s$ using the 2-approximation algorithm from Theorem 58.1.

(3) By Lemma 58.10, T is a $(k-1)$-fragment transversal of $H = H_s - s$.
Find a forest F on T as in Corollary 58.1 so that $G = H + F$ is k-connected.

To bound the approximation ratio we use the following technical statement.

Lemma 58.11

Let T be a node set with node weights $w(v) \geq 0$, $v \in T$. If F is a forest on T then

$$\sum_{uv \in F} (w(u) + w(v)) \leq (|T| - 2) \max_{v \in T} w(v) + \sum_{v \in T} w(v)$$

The approximation ratio follows from the following two lemmas.

Lemma 58.12

$c(H_s) \leq 2\mathsf{opt}$ for the graph H_s computed at step 2 of the algorithm.

Proof

Let G^* be an optimal k-connected spanning subgraph of $\mathcal{G}$. Extend G^* to a spanning subgraph G_s^* by adding to G^* the node s together with edge set $\delta_{\mathcal{G}_s}(s)$. It is easy to see that G_s^* is k-outconnected from s, and clearly $c(G_s^*) = c(G^*)$. Thus $c(H_s) \leq 2c(G_s^*) = 2c(G^*) = 2\mathsf{opt}$. □

Lemma 58.13

$c(F) \leq \frac{k-1}{n}\mathsf{opt}$ holds for the forest F computed at step 3 of the algorithm.

Proof

Denote $w_0 = w(v_0) = 0$ and $w_i = w(v_i) = c(v_0 v_i)$, $i = 1, \dots, k$, where $w_1 \leq w_2 \leq \cdots \leq w_k$. Since the costs are metric, $c(v_i v_j) \leq w_i + w_j$, $0 \leq i \neq j \leq k$. We claim that $c(J_{k+1}) = w_k + \sum_{v \in T} w(v) \leq \frac{2}{n}\mathsf{opt}$. Indeed, if G^* is an optimal k-connected spanning subgraph of $\mathcal{G}$, then $\sum\{c(\delta_{G^*}(v)) : v \in V\} = 2c(G^*) = 2\mathsf{opt}$, and thus there is a node v with $c(J_{k+1}(v)) \leq c(\delta_{G^*}(v)) \leq \frac{2}{n}\mathsf{opt}$. By our choice of J_{k+1}, $w_k + w_{k-1} \leq 2\mathsf{opt}/n$, thus $w_{k-1} = \max\{w(v) : v \in T\} \leq \frac{1}{n}\mathsf{opt}$. Using this, the metric costs assumption, and Lemma 58.11 we get

$$
\begin{aligned}
c(F) \leq & \sum_{v_i v_j \in F} (w_i + w_j) \leq (k-2)w_{k-1} + \sum_{v \in T} w(v) \leq (k-3)w_{k-1} \\
& + \frac{2}{n}\mathsf{opt} \leq \frac{k-3}{n}\mathsf{opt} + \frac{2}{n}\mathsf{opt} = \frac{k-1}{n}\mathsf{opt}
\end{aligned}
$$

□

58.6.2 Directed Graphs

A $v \rightarrow \ell$*-star* is a directed tree rooted at v, with ℓ nodes and $\ell - 1$ leaves; a $v \leftarrow \ell$*-star* is a graph where reversal of its edges results in a $v \rightarrow \ell$-star. Among all subdigraphs of $\mathcal{G}$ which are $v \rightarrow \ell$-stars (resp., $v \leftarrow \ell$-stars), let $J_\ell^-(v)$ (resp., $J_\ell^+(v)$) be a cheapest one. The algorithm for directed graphs is as follows:

(1) Find a node v_0 for which $c(J_{k+1}^-(v)) + c(J_{k+1}^+(v))$ is minimal, and set $u_0 = v_0$.
Let $\{v_1, \dots, v_k\}$ be the leaves of $J_{k+1}^- = J_{k+1}^-(v_0)$, and $T^+ = \{u_1, \dots, u_k\}$ be the leaves of $J_{k+1}^+ = J_{k+1}^+(u_0)$, where $c(v_0 v_i) \leq c(v_0 v_{i+1})$ and $c(u_i u_0) \leq c(u_{i+1} u_0)$, $i = 1, \dots, k-1$.

(2) Set $T^- = \{v_0, \dots, v_{k-1}\}$ and $T^+ = \{u_0, \dots, u_{k-1}\}$. Add a node s to $\mathcal{G}$ and edges $v_i s$, $s u_i$ of the cost 0, $i = 0, \dots, k-1$, obtaining a graph $\mathcal{G}_s$. Compute two spanning subgraphs of $\mathcal{G}_s$: an optimal k-outconnected from s, say H_s^-, and an optimal k-inconnected to s, say H_s^+.

(3) By Lemma 58.10 (T^-, T^+) is a $(k-1)$-fragment transversal of $H = (H_s^- + H_s^+) - s$.
Find an edge set $F \subseteq \delta_{\mathcal{G}}(T^-, T^+)$ without alternating cycles so that $G = H + F$ is k-connected.

The approximation ratio follows from the following directed counterparts of Lemmas 58.11, 58.12, and 58.13.

Lemma 58.14

Let A, B be disjoint node sets with nonnegative weights $w(v) \geq 0$, $v \in A + B$. If F is a forest on $A + B$ so that every edge in F connects a node in A to a node in B then

$$
\sum_{ab \in F} (w(a) + w(b)) \leq (|B| - 1)\max_{a \in A} w(a) + (|A| - 1)\max_{b \in B} w(b) + \sum_{v \in A+B} w(v)
$$

Lemma 58.15

$c(H) \leq c(H_s^-) + c(G_s^+) \leq 2\mathsf{opt}$.

Lemma 58.16

$c(F) \leq \frac{k}{n}\mathsf{opt}$.

58.7 General Costs

58.7.1 A 2-Approximation Algorithm for Undirected Edge-GSN

The crucial property used by the algorithm of Ref. [25] is as follows.

Theorem 58.16 (Jain [25])

For a skew-supermodular p, any basic solution x of Eq. (58.9) *has an entry of value $x_e \geq 1/2$.*

Given Theorem 58.16, a 2-approximation algorithm immediately follows. As long as G_0 is not a feasible solution (initially $G_0 = (V, \emptyset)$), we repeatedly find a basic optimal solution x to linear program (58.9) (for p defined by Eq. (58.8) this can be done in polynomial time) and transfer the edge e with $x_e \geq 1/2$

from $I = \mathcal{E} - E_0$ to G_0. Every iteration reduces the optimum of linear program (58.9) by at least $c_e/2$, and increases the cost of G_0 by c_e. Hence, the total cost of the solution over all iteration is at most twice the initial optimum of linear program (58.9). We prove a weaker version of Theorem 58.16; Theorem 58.16 can be proved by a slight refinement using parity arguments.

Claim 58.1

For a skew-supermodular p, any basic solution x of linear program (58.9) *has an entry of value $x_e \geq 1/3$.*

Proof

Assume $0 < x_e < 1$ for all $e \in I$ (for $e \in I$, if $x_e = 0$ then e can be ignored, and if $x_e = 1$ then we are done). By Theorem 58.6, there exists an x-defining family $\mathcal{L}$ which is laminar.

To get a contradiction, assume that the statement is false. With every $e = uv \in I$ associate two endpoints $e_u \sim u$ and $e_v \sim v$. The number of endpoints is thus $2m$, where $m = |I|$. For $A \in \mathcal{L}$, let $\mathcal{L}_A = \{X \in \mathcal{L} : X \subseteq A\}$. Under the assumption that $x_e < 1/3$ for all e, we show that we are able to do the following. Given $A \in \mathcal{L}$, we can assign every endpoint contained in A to a set in $\mathcal{L}_A$ (every endpoint is assigned to exactly one set) such that: A gets four endpoints, and any other set in $\mathcal{L}_A$ gets at least two endpoints. This implies that we are able to assign at least $2m + 2 > 2m$ distinct endpoints to sets in $\mathcal{L}$, a contradiction.

The proof is by induction on $|\mathcal{L}_A|$. The induction basis is $|\mathcal{L}_A| = 1$. In this case, $d_I(A) \geq 4$, since $p(A) \geq 1$ and since $x_e \leq 1/3$ for every $e \in \delta_I(A)$. Thus we can assign to A the four endpoints that belong to A of the edges in $\delta_I(A)$ (these may be four “copies” of the same node).

Henceforth, assume that $|\mathcal{L}_A| \geq 1$. Let us say that B is a *child of* A if B is a maximal inclusion set in $\mathcal{L}$ properly contained in A. By the induction hypothesis, for any child B of A, in $\mathcal{L}_B$ we can assign the endpoints contained in B such that: B gets four endpoints, and any other set in $\mathcal{L}_B$ gets at least two endpoints. In particular, we can assign the endpoints contained in A such that: A gets zero endpoints, every child of A gets four endpoints, and any other set in $\mathcal{L}_A$ gets at least two endpoints. If A has at least two children, then by transferring two endpoints from every child of A to A, we get an assignment as claimed.

The remaining case is when A has a unique child B. We again move two extra endpoints of B to A. We show that there are at least two endpoints in $A - B$, which we assign to A. If there is no endpoint in $A - B$ then $\delta_I(A) = \delta_I(B)$ and thus the equations corresponding to A and B are the same. This contradicts that $\mathcal{L}$ is x-defining. $A - B$ cannot contain exactly one endpoint, since there is an edge $e \in I$ so that either $\delta_I(A) = \delta_I(B) + e$ or $\delta_I(B) = \delta_I(A) + e$. Since A, B are tight this implies $|r(A) - r(B)| = x_e$, which is a contradiction, since $|r(A) - r(B)|$ is an integer. □

58.7.2 Approximation Algorithm for *k*-CSS

Theorem 58.17 (Kortasarz and Nutov [31], Cheriyan, Vempala and Vetta [30])

For k-CSS there exists an $O(\frac{n}{n-k} \ln^2 k)$-approximation algorithm for both directed and undirected graphs, and an $O(\ln k)$-approximation algorithm for undirected graphs with $n \geq 2k^2$.

The proof of Theorem 58.17 follows. The algorithm has k iterations. For $\ell = 0, \ldots, k-1$, iteration ℓ starts with an already computed ℓ-connected spanning subgraph $G = (V, E)$ of $\mathcal{G}$ (so edges of G have cost zero), finds an augmenting edge set F such that $G + F$ is $(\ell + 1)$-connected, and adds F to G.

We say that $U \subseteq V$ is an *ℓ-cover of* G if no ℓ-separator of G contains U, that is, if U intersects the set $V - C$ for every ℓ-separator C of G. Let U be an ℓ-cover of G. Using the algorithm of Ref. [36] we find an augmenting edge set F so that $G + F$ is $(\ell + 1)$-connected of cost $c(F) \leq 2|U|\mathsf{opt}$, as follows. For every $s \in U$ compute an edge set F_s of cost $\leq 2\mathsf{opt}$ such that $G + F_s$ is $(\ell + 1)$-outconnected from s (and also $(\ell + 1)$-inconnected to s, for directed graphs), and set $F = \cup_{s \in U} F_s$ to be the union of the computed edge sets. Note however that Lemma 58.1 implies that $c(F_s) \leq \frac{2|U|}{k-\ell}\mathsf{opt}_k$, where opt_k is the optimal value of LP-relaxation (58.7) with $p(X', X'') = k - (n - |X' + X''|)$. Thus for both directed and undirected graphs the following holds.

Proposition 58.2

Suppose that there is a polynomial algorithm that finds in any ℓ-connected graph G on n nodes an ℓ-cover of G of size at most $t(\ell, n)$. Then there exists a polynomial-time algorithm that for instances of the minimum k-connected spanning subgraph problem on n nodes finds a feasible solution of cost at most $\mathsf{opt}_k \cdot 2\sum_{\ell=0}^{k-1} \frac{t(\ell,n)}{k-\ell} = \mathsf{opt}_k \cdot O(\ln k \cdot \max_{0\le\ell\le k-1} t(\ell, n))$.

Theorem 58.1 follows by combining Proposition 58.2 with the following two theorems from Refs. [46] and [31], respectively.

Theorem 58.18 **(Mader [46])**

Any undirected ℓ-connected graph G with $n \geq 2\ell^2$ nodes has an ℓ-cover of size 3.

Theorem 58.19 **(Kortasarz and Nutov [31])**

There exists a polynomial algorithm that given an ℓ-connected (directed or undirected) graph G with $n \geq \ell+2$ nodes finds an ℓ-cover of G of size $O\left(\frac{n}{n-\ell} \ln \ell\right)$.

58.8 Hardness of Approximation: Three Typical Reductions

We illustrate three typical reductions for establishing approximation hardness of connectivity problems.

3-Partition (strongly NP-complete [47], used for proving NP-hardness)

Instance: A set $\mathcal{A} = \{\alpha_1, \ldots, \alpha_{3m}\}$ of positive integers so that $\beta = \sum_{i=1}^{3m} \alpha_i / m$ is an integer, and so that $\beta/4 \leq \alpha < \beta/2$ for every $\alpha \in \mathcal{A}$.
Question: Can $\mathcal{A}$ be partitioned in to m sets $\mathcal{A}_1, \ldots, \mathcal{A}_m$ so that each set sums to exactly β?

Kant and Bodlaender [48] were the first to use the type of reductions described below for establishing that the problem of augmenting a 1-connected graph to be 2-connected while preserving planarity is NP-hard. We will describe a version from [32].

Theorem 58.20 **(Nagamochi and Ishii [19])**

The undirected node-connectivity augmentation problem with $r(u, v) \in \{0, 2\}$ is NP-hard.

Proof

3-Partition is strongly NP-complete [47]. Therefore, it is enough to show a pseudopolynomial time reduction from 3-partition to the problem in the theorem. Note that if the answer to 3-partition is "YES," then each $\mathcal{A}_i$ contains exactly three elements from A. We can assume that $\alpha \geq 3$ for every $\alpha \in \mathcal{A}$; otherwise, we get an equivalent instance by increasing each $\alpha \in \mathcal{A}$ by 2.

Here is the reduction. Given an instance $\mathcal{A} = \{\alpha_1, \ldots, \alpha_{3m}\}$ of 3-partition, construct an instance $(G_0 = (V, E_0), r)$ of the undirected node-connectivity augmentation problem with $r(u, v) \in \{0, 2\}$ as follows. Set $V = A + B + \{b_1, \ldots, b_m\} + \{s\}$ where $|A| = |B| = m\beta$. Partition A into $3m$ sets $A_1, \ldots, A_{3m}$ where $|A_i| = \alpha_i$ for $i = 1, \ldots, 3m$, and partition B into m sets $B_1, \ldots, B_m$ of size β each. Let $E_0 = \{vs : s \in V - B\} + \cup_{i=1}^{m} \{b_i v : v \in B_i\}$. The requirement function is defined by $r(u, v) = 2$ if u, v belong to the same part A_i of A or to the same part B_j of B, and $r(u, v) = 0$ otherwise. We claim that the answer to 3-partition is "YES" if, and only if, (G_0, r) has a solution F of size $m\beta$.

Suppose that the answer to 3-partition is "YES," and that the corresponding parts of $\mathcal{A}$ are $\mathcal{A}_i = \{a_{3i-2}, a_{3i-1}, a_{3i}\}$, $i = 1, \ldots, m$. Let F_i be an arbitrary perfect matching between $A_{3i-2} + A_{3i-1} + A_{3i}$ and B_i. It is easy to see that $F = \cup_{i=1}^{m} F_i$ is a feasible solution for (G_0, r) of size $|F| = m\beta$.

If (G_0, r) has a feasible solution F of size $m\beta$, then F must be a perfect matching on $A + B$. Let $a', a'' \in A_i$ for some i. By the definition of r, $H = (G + F) - s$ has a uv-path for any $u, v \in A_i$. Note that $a'a'' \notin F$, as otherwise for $a \in A_i - \{a', a''\}$ (such a exists since $|A_i| \geq 3$) there cannot be a an aa'-path in H. Let $a'b', a''b'' \in F$. It is easy to see that then there is an $a'a''$-path in $(G + F) - s$ if, and only if, b', b'' belong to the same part B_j of B. It follows therefore that for any part A_i of A there exists a part B_j

of B so that $\Gamma_F(A_i) \subseteq B_j$. Define $\mathcal{A}_j = \{a_i : \Gamma_F(A_i) \subseteq B_j\}$. This gives a solution for the 3-partition instance. □

Set-Cover (cannot be approximated within $C \ln n$ for some universal constant $C < 1$ even on instances with $|A| = |B|$, unless P = NP [49]).

Instance: A bipartite graph $J = (A + B, I)$ without isolated nodes.
Objective: Find a minimum size subset $T \subseteq A$ such that $\Gamma_J(T) = B$.

Theorem 58.21 (Frank [19])

*The directed rooted edge-*GSN *augmentation (the case of* $\{0, 1\}$*-costs) with* $r(u, v) \in \{0, 1\}$ *cannot be approximated within* $C \ln n$ *for some universal constant* $C < 1$*, unless* P=NP.

Proof
Given an instance $J = (A + B, I)$ for set-cover (with $|A| = |B|$) construct an instance $G_0 = (V, E_0)$ for directed rooted edge-GSN augmentation by directing the edges in J from A to B, adding a new node s, and setting $r(s, v) = 1$ for every $v \in B$. Let F be a feasible solution for (G_0, r) and let $e = uv \in F$. If $u \neq s$ then we replace e by the link sv, getting again a feasible solution. Then, if $e = sv$ and $v \in B$, we replace e by a link uv' where $v' \in \{a \in A : v \in \Gamma(a)\}$ (such v' exists, since J has no isolated nodes), getting again a feasible solution. This implies that for any solution F' for the the obtained instance (G_0, r) there exists a solution F with $|F| = |F'|$ such that every edge in F connects s to some node in $V - B = A$. But for such F, $T \subseteq A$ is a solution for set-cover on J if, and only if, $F = \{sv : v \in T\}$ is a solution for (G_0, r). Combined with the hardness result from Ref. [49], we get the statement. □

MinRep (cannot be approximated within $O(2^{\log^{1-\varepsilon} n})$ for any $\varepsilon > 0$, unless NP $\subseteq$ DTIME$(n^{\text{polylog}(n)})$)

Instance: A bipartite graph $H = (A + B, I)$, and equitable partitions $\mathcal{A}$ of A and $\mathcal{B}$ of B.
Objective: Find a minimum size node set $A' \cup B'$, where $A' \subseteq A$ and $B' \subseteq B$, so that for any $A_i \in \mathcal{A}$, $B_j \in \mathcal{B}$ with $\delta_I(A_i, B_j) \neq \emptyset$ there are $a \in A' \cap A_i$, $b \in B' \cap B_j$ such that $ab \in I$.

Theorem 58.22 (Raz [50])

MinRep on n *nodes cannot be approximated within* $O(2^{\log^{1-\varepsilon} n})$ *for any* $\varepsilon > 0$*, unless* NP $\subseteq$ DTIME $(n^{\text{polylog}(n)})$.

Theorem 58.23 (Dodis and Khanna [26])

*The directed edge-*GSN *with cost in* $\{0, 1, \infty\}$ *and* $r(u, v) \in \{0, 1\}$ *cannot be approximated within* $O(2^{\log^{1-\varepsilon} n})$ *for any fixed* $\varepsilon > 0$*, unless* NP $\subseteq$ DTIME$(n^{\text{polylog}(n)})$.

Proof
Given an instance $(H = (A + B, I), \mathcal{A}, \mathcal{B})$ of MinRep construct an instance $(\mathcal{G} = (V, E_0 + E_1), r)$ of directed edge-GSN as follows, where edges in E_0 have cost 0 and edges in E_1 have cost 1. Let $\mathcal{I} = \{ij : A_i \in \mathcal{A}, B_j \in \mathcal{B}, \delta_H(A_i, B_j) \neq \emptyset\}$. The graph $\mathcal{G} = (V, E_0 + E_1)$ is obtained from H as follows:

1. Add to H: a set $\{a_1, \ldots, a_{|\mathcal{A}|}, b_1, \ldots, b_{|\mathcal{B}|}\}$ of $|\mathcal{A}| + |\mathcal{B}|$ nodes, and for every $ij \in \mathcal{I}$ a pair of nodes a_{ij}, b_{ij} (so a total number of nodes added to H is $|\mathcal{A}| + |\mathcal{B}| + 2|\mathcal{I}|$). Thus

$$V = A + B + \{a_1, \ldots, a_{|\mathcal{A}|}, b_1, \ldots, b_{|\mathcal{B}|}\} + \{a_{ij} : ij \in \mathcal{I}\} + \{b_{ij} : ij \in \mathcal{I}\}$$

2. For every $ij \in \mathcal{I}$: connect a_{ij} to a_i and connect b_j to b_{ij}. Thus:

$$E_0 = I + \{a_{ij}a_i : ij \in \mathcal{I}\} + \{b_j b_{ij} : ij \in \mathcal{I}\}$$

The edges that can be added by cost 1 each are from a_i to A_i or from B_j to b_j, that is:

$$E_1 = \{a_i a : a \in A_i \in \mathcal{A}\} + \{bb_j : b \in B_j \in \mathcal{B}\}$$

The requirement function is defined by $r(a_{ij}, b_{ij}) = 1$ for $ij \in \mathcal{I}$ and $r(u, v) = 0$ otherwise.

We claim that an edge set $F \subseteq E_1$ is a feasible solution for $(\mathcal{G}, r)$ if, and only if, the end-nodes of F contained in $A + B$ are a feasible solution for the original MinRep instance. Note that there is a bijective correspondence between edge sets $F \subseteq E_1$ and subsets $A' + B'$ of $A + B$, where $A' \subseteq A$, $B' \subseteq B$. Namely

$$F = \{a_i a : a \in A_i, 1 \leq i \leq |\mathcal{A}|\} \cup \{b_j b : b \in B_j, 1 \leq j \leq |\mathcal{B}|\}$$

Let $A' + B'$ and F be such corresponding pair. Recall that $A' + B'$ is a feasible solution for MinRep if for every $ij \in \mathcal{I}$ there are $a \in A' \cap A_i$, $b \in B' \cap B_j$ such that $ab \in I$. Note that for $ij \in \mathcal{I}$ there are such a, b if, and only if, there is an $a_{ij}b_{ij}$-path a_{ij}, a_i, a, b, b_j, b_{ij} of the length 5 in $(V, E_0 + F)$.

Since in the construction $|V| = O(n^2)$, where $n = |A| + |B|$, Theorem 58.22 implies Theorem 58.23.

□

Theorem 58.23 easily extends to the case of $\{1, \infty\}$-costs [44], and to metric costs (using metric completion). For an extension to undirected graphs and to $\{0, 1\}$-costs of Theorems 58.21 and 58.23 see Refs. [34] and [7,27], respectively.

58.9 Open Problems

- Determining the complexity status of the two undirected problems: augmenting a k-connected graph to be $(k+1)$-connected, and augmenting a k-outconnected graph to be $(k+1)$-outconnected.
- Can one achieve a constant approximation ratio for *undirected* CA with $r_{\max}$ bounded by a constant? Can one a achieve an approximation ratio $O(n^{1-\varepsilon})$ for node CA?
- Improving hardness results or approximation ratios for k-CSS.
- Can one achieve an approximation ratio $2 - \varepsilon$ for k-ECSS? (This is open even if $k = 2$ and $\mathcal{G}$ contains a spanning tree of cost zero.)
- Can one achieve an approximation ratio better than 3/2 (e.g., 4/3) for metric 2-ECSS?

References

[1] Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Scrijver, A., *Combinatorial Optimization*, Wiley, New York, 1998.

[2] Karpinski, M. and Zelikovsky, A., New approximation algorithms for the Steiner tree problem, *J. Comb. Optim.*, 1, 47, 1997.

[3] Charikar, M., Chekuri, C., Cheung, T., Dai, Z., Goel, A., Guha, S., and Li, M., Approximation algorithms for directed Steiner problems, *J. Algorithms*, 33, 73, 1999.

[4] Benczúr, A. and Frank, A., Covering symmetric supermodular functions by graphs, *Math. Prog.*, 84, 483, 1999.

[5] Cheriyan, J., Vempala, S., and Vetta, A., Network design via iterative rounding of setpair relaxations, *Combinatorica*, 26(3), 255, 2006.

[6] Fleischer, L., Jain, K., and Williamson, D. P., An iterative rounding 2-approximation algorithm for the element connectivity problem, *Proc. FOCS*, 2001, p. 339.

[7] Nutov, Z., Approximating connectivity augmentation problems, *Proc. SODA*, 2005, p. 176.

[8] Frank, A., Edge-connection of graphs, digraphs, and hypergraphs, EGRES TR No 2001-11, 2001.

[9] Auletta, V., Dinitz, Y., Nutov, Z., and Parente, D., A 2-approximation algorithm for finding an optimum 3-vertex-connected spanning subgraph, *J. Algorithms*, 32(1), 21, 1999.

[10] Fredrickson, G. N. and Jájá, J., On the relationship between the biconnectivity augmentation and traveling salesman problem, *Theor. Comp. Sci.*, 19(2), 189, 1982.

[11] Jothi, R., Raghavachari, B., and Varadarajan, S., A 5/4-approximation algorithm for minimum 2-edge-connectivity, *Proc. SODA*, 2003, p. 725.

[12] Kortsarz, G. and Nutov, Z., Approximating node connectivity problems via set covers, *Algorithmica*, 37, 75, 2003.

[13] Khuller, S., Raghavachari, B., and Young, N. E., Approximating the minimum equivalent digraph, *SIAM J. Comput.*, 24(4), 859, 1995.

[14] Vempala, S. and Vetta, A., Factor 4/3 approximations for minimum 2-connected subgraphs, *Workshop on Approximation Algorithms*, 2000, p. 262.

[15] Khuller, S., Approximation algorithms for finding highly connected subgraphs, in *Approximation Algorithms for NP-Hard Problems,* Hochbaum, D. S., Ed., PWS, Boston, 1995, chap. 6.

[16] Frank, A., Connectivity and network flows, in *Handbook of Combinatorics,* Graham, R. L., Grötschel, M., and Lovász, L., Eds., Elsevier, Amsterdam, 1995, chap. 2.

[17] Frank, A., Connectivity augmentation problems in network design, *Math. Programming: State of the Art*, 34, 84, 1995.

[18] Watanabe, T. and Nakamura, A., Edge-connectivity augmentation problems, *Comput. Syst. Sci.*, 35(1), 96, 1987.

[19] Frank, A., Augmenting graphs to meet edge-connectivity requirements, *SIAM J. Disc. Math.*, 5(1), 25, 1992.

[20] Frank, A. and Jordán, T., Minimal edge-coverings of pairs of sets, *J. Comb. Theor. B*, 65, 73, 1995.

[21] Jackson, B. and Jordán, T., A near optimal algorithm for vertex connectivity augmentation, *Symp. on Algorithms and Computation*, 2000, p. 313.

[22] Kortsarz, G. and Nutov, Z., Tight approximation for connectivity augmentation problems, in Proc. *ICALP*, 2006, p. 443.

[23] Gabow, H. N., Goemans, M. X., Tardos, E., and Williamson, D. P., Approximating the smallest k-edge connected spanning subgraph by LP-rounding, *Proc. SODA*, 2005, p. 562.

[24] Cheriyan, J. and Thurimella, R., Approximating minimum-size k-connected spanning subgraphs via matching, *SIAM J. Comput.*, 30(2), 528, 2000.

[25] Jain, K., A factor 2 approximation algorithm for the generalized Steiner network problem, *Combinatorica*, 21(1), 39, 2001.

[26] Dodis, Y. and Khanna, S., Design networks with bounded pairwise distance, *Proc. STOC*, 1999, p. 750.

[27] Kortsarz, G., Krauthgamer, R., and Lee, J. R., Hardness of approximation for vertex-connectivity network design problems, *SIAM J. Comput.*, 33(3), 704, 2004.

[28] Khuller, S. and Vishkin, U., Biconnectivity approximations and graph carvings, *JACM*, 41(2), 214, 1994.

[29] Cheriyan, J. and Vetta, A., Approximation algorithms for network design with metric costs, *SIAM J. Discr. Math.* (to appear). Also in *Proc. STOC*, 2005, p. 167.

[30] Cheriyan, J., Vempala, S., and Vetta, A., An approximation algorithm for the minimum-cost k-vertex connected subgraph, *SIAM J. Comput.*, 32(4), 1050, 2003.

[31] Kortsarz, G. and Nutov, Z., Approximation algorithm for k-node connected subgraphs via critical graphs, *SIAM J. Comput.* (to appear). Also in *Proc. STOC*, 2004, p. 138.

[32] Nagamochi, H. and Ishii, T., On the minimum local-vertex-connectivity augmentation in graphs, *Disc. Appl. Math.*, 129(2–3), 475, 2003.

[33] Jackson, B. and Jordán, T., Independence free graphs and vertex connectivity augmentation, *J. Comb. Theor. B*, 94, 31, 2005.

[34] Nutov, Z., Approximating rooted connectivity augmentation problems, *Algorithmica*, 44(3), 213, 2006.

[35] Edmonds, J., Matroid intersection, *Ann. Disc. Math.*, 4, 39, 1979.

[36] Frank, A. and Tardos, É., An application of submodular flows, *Linear Algebra Appl.*, 114/115, 329, 1989.

[37] Cheriyan, J., Jordán, T., and Nutov, Z., On rooted node-connectivity problems, *Algorithmica*, 30(3), 353, 2001.

[38] Jordán, T., On the optimal vertex-connectivity augmentation, *J. Comb. Theor. B*, 63, 8, 1995.

[39] Mader, W., Ecken vom grad n in minimalen n-fach zusammenhängenden graphen, *Archive der Mathematik*, 23, 219, 1972.

[40] Mader, W., Minimal n-fach in minimalen n-fach zusammenhängenden digraphen, *J. Comb. Theor. B*, 38, 102, 1985.

[41] Melkonian, V. and Tardos, E., Algorithms for a network design problem with crossing supermodular demands, *Networks*, 43(4), 256, 2004.

[42] Wolsey, L. A., An analysis of the greedy algorithm for the submodular set covering problem, *Combinatorica*, 2, 385, 1982.

[43] Jordán, T., A note on the vertex connectivity augmentation, *J. Comb. Theor. B*, 71(2), 294, 1997.

[44] Liberman, G. and Nutov, Z., On shredders and vertex-connectivity augmentation, *J. Disc. Algorithms*, 5, 91, 2007.

[45] Khuller, S. and Raghavachari, B., Improved approximation algorithms for uniform connectivity problems, *J. Algorithms*, 21, 434, 1996.

[46] Mader, W., On k-con-critically n-connected graphs, *J. Comb. Theor. B*, 86(2), 296, 2002.

[47] Garey, M. R. and Johnson, D. S., *Computers and intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[48] Kant, G. and Bodlaender, H. L., Planar graph augmentation problems, *Workshop on Disc. Algorithms*, 1991, p. 286.

[49] Raz, R. and Safra, S., A sub-constant error-probability low-degree test and a sub-constant error-probability PCP characterization of NP, *Proc. STOC*, 1997, p. 475.

[50] Raz, R., A parallel repitition theorem, *SIAM J. Comput.*, 27(3), 763, 1998.

59

Optimum Communication Spanning Trees

Bang Ye Wu
Shu-Te University

Chuan Yi Tang
National Tsing Hua University

Kun-Mao Chao
National Taiwan University

59.1 Introduction

A spanning tree for a graph G is a subgraph of G that is a tree and contains all the vertices of G. Besides numerous network design applications, spanning trees also play important roles in several newly established research areas, such as biological sequence alignments and evolutionary tree construction. In contrast, there exists many spanning tree problems that have been proved to be NP-hard. Thus, designing approximation algorithms for those hard spanning tree problems has become an exciting and important field in theoretical computer science. This chapter focuses on the approximation algorithms for constructing efficient communication spanning trees.

Let $G = (V, E, w)$ be an undirected graph with nonnegative edge length function w and $\lambda(u, v)$ the requirements for each pair of vertices. The *optimum communication spanning tree* (OCT) problem is defined as follows. For any spanning tree T of G, the communication cost between two vertices is defined to be the requirement multiplied by the path length of the two vertices on T, and the communication cost of T is the total communication cost summed over all pairs of vertices. Our goal is to construct a spanning tree with minimum communication cost. That is, we want to find a spanning tree T such that $\sum_{u,v \in V} \lambda(u, v) d_T(u, v)$ is minimized.

The requirements in the OCT problem are arbitrary nonnegative values. By restricting the requirements, several special cases of the problem were defined in the literature. In the following, we compile a list of the problems, in which $r : V \to Z_0^+$ is a given vertex weight function, and $S \subset V$ a set of sources:

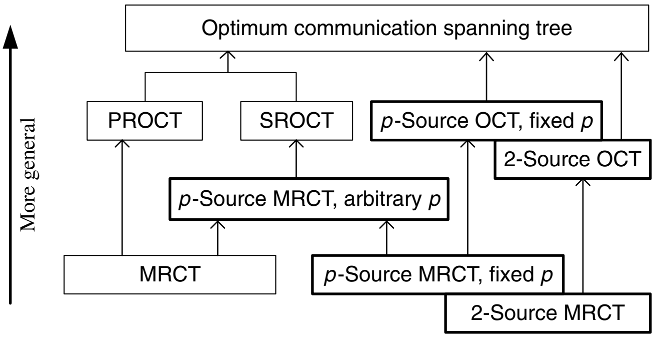


FIGURE 59.1 The relationships between OCT problems [8].

TABLE 59.1 The Restrictions and Currently Best Ratios of the OCT Problems

Problem	Restriction on Requirements	Ratio	Reference
MRCT	$\lambda(u, v) = 1$	PTAS	[1,10,11]
PROCT	$\lambda(u, v) = r(u)r(v)$	PTAS	[12,13]
SROCT	$\lambda(u, v) = r(u) + r(v)$	2	[12]
p-MRCT	$\lambda(u, v) = r(u) + r(v)$	2	[12]
	$r(s) = 1$ for $s \in S$, and $r(v) = 0$ otherwise		
2-MRCT	The same as p-MRCT but $\lvert S\rvert = 2$	PTAS	[14]
Weighted	The same as 2-MRCT but $r(s_1) \neq r(s_2)$	2 (general graphs)	[14]
2-MRCT		PTAS (metric graphs)	[14]
p-OCT	$\lambda(u, v) = 0$ for $u, v \notin S$ and $\lvert S\rvert = p$ is a constant	2 (metric graphs)	[15]
2-OCT	$\lambda(u, v) = 0$ for $u, v \notin S$, $\lvert S\rvert = 2$	3 (general graphs)	[15]

- $\lambda(u, v) = 1$ for each $u, v \in V$: This version is called the Minimum Routing Cost Spanning Tree (MRCT) problem.
- $\lambda(u, v) = r(u)r(v)$ for each $u, v \in V$: This version is called the Optimum Product-Requirement Communication Spanning Tree (OPRCT) problem.
- $\lambda(u, v) = r(u) + r(v)$ for each $u, v \in V$: This version is called the Optimum Sum-Requirement Communication Spanning Tree (OSRCT) problem.
- $\lambda(u, v) = 0$ if $u \notin S$: This version is called the p-Source OCT (p-OCT) problem. In other words, the goal is to find a spanning tree minimizing $\sum_{u\in S}\sum_{v\in V}\lambda(u, v)d_T(u, v)$.
- $\lambda(u, v) = 1$ if $u \in S$, and $\lambda(u, v) = 0$ otherwise: This version is called the p-source MRCT (p-MRCT) problem. In other words, the goal is to find a spanning tree minimizing $\sum_{u\in S}\sum_{v\in V} d_T\,(u, v)$.

Figure 59.1 depicts the relationships between the problems, and Table 59.1 gives the currently best approximation ratio of each problem.

The rest of the chapter is organized as follows. Section 59.2 presents some approximation schemes for the MRCT problem. We give a PTAS for the PROCT problem in Section 59.3, and a 2-approximation algorithm for the SROCT problem in Section 59.4. Sections 59.5 and 59.6 propose some approximation algorithms for the multiple-sources MRCT and OCT problems, respectively. Finally, Section 59.7 concludes the chapter with a few remarks.

59.2 Minimum Routing Cost Spanning Tree

In the MRCT problem, the requirements between any pair of vertices are the same. In other words, we want to find the spanning tree minimizing the all-to-all distance. When there is no ambiguity, we assume that $G = (V, E, w)$ is the given graph, which is simple, connected, and undirected. In this section, $\widehat{T}$ denotes an MRCT of G, and $c(T)$ is the routing cost of a spanning tree T.

59.2.1 Shortest-Paths Trees and Solution Decomposition

Let r be the *median* of graph G, that is, the vertex with minimum total distance to all vertices, and Y any shortest-paths tree rooted at r. By the triangle inequality, we have $d_Y(u, v) \leq d_Y(u, r) + d_Y(v, r)$ for any vertices u and v. Summing up over all pairs of vertices, we obtain that $c(Y) \leq 2n \sum_v d_Y(v, r)$. Since r is a median, $\sum_v d_G(r, v) \leq \sum_v d_G(u, v)$ for any vertex u, it follows $\sum_v d_G(r, v) \leq (1/n) \sum_{u,v} d_G(u, v)$. By the property of a shortest-paths tree, $d_Y(r, v) = d_G(r, v)$ for each vertex v, and consequently, $c(Y) \leq 2n \sum_v d_G(r, v) \leq 2 \sum_{u,v} d_G(u, v)$. Since $c(\widehat{T}) \geq \sum_{u,v \in V} d_G(u, v)$, we have that Y is a 2-approximation of an MRCT.

Theorem 59.1 (**Wong [1]**)

A shortest-paths tree rooted at the median of a graph is a 2-approximation of an MRCT of the graph.

The median of a graph can be found easily once the distances of all pairs of vertices are known. By Theorem 59.1, we have a 2-approximation algorithm and the time complexity is dominated by that of finding all-pairs shortest path lengths of the input graph.

Corollary 59.1

An MRCT of a graph can be approximated with ratio 2 in $O(n^2 \log n + mn)$ *time.*

Now we introduce another proof of the approximation ratio of the shortest-paths tree. The analysis technique we used is called *solution decomposition*, which is widely used in algorithm design, especially for approximation algorithms. To design an approximation algorithm for an optimization problem, we first suppose that X is an optimal solution. Then we decompose X and construct another feasible solution Y. To our aim, Y is designed to be a good approximation of X and belongs to some restricted class of feasible solutions, of which the best solution can be found efficiently. The algorithm is designed to find an optimal solution of the restricted problem, and the approximation ratio is ensured by that of Y. It should be noted that Y plays a role only in the analysis of approximation ratio, but not in the designed algorithm. In the following, we show how to design a 2-approximation algorithm by this method.

For any tree, we can always cut it at a node r such that each branch contains at most half of the nodes. Such a node is usually called a *centroid* of the tree in the literature. Suppose that r is the centroid of the MRCT $\widehat{T}$. If we construct a shortest-paths tree Y rooted at the centroid r, the routing cost will be at most twice that of $\widehat{T}$. This can be easily shown as follows. First, if u and v are two nodes not in a same branch, $d_{\widehat{T}}(u, v) = d_{\widehat{T}}(u, r) + d_{\widehat{T}}(v, r)$. Consider the total distance of all pairs of nodes on $\widehat{T}$. For any node v, since each branch contains no more than half of the nodes, the term $d_{\widehat{T}}(v, r)$ will be counted in the total distance at least n times, $n/2$ times for v to others and $n/2$ times for others to v. Hence, we have $c(\widehat{T}) \geq n \sum_v d_{\widehat{T}}(v, r)$. Since, as in the proof of Theorem 59.1, $c(Y) \leq 2n \sum_v d_G(v, r)$, it follows that $c(Y) \leq 2c(\widehat{T})$. We have decomposed the optimal solution $\widehat{T}$ and constructed a 2-approximation Y. Of course, there is no way to know what Y is since the optimal $\widehat{T}$ is unknown. But we have the next result.

Lemma 59.1

There exists a vertex such that any shortest-paths tree rooted at the vertex is a 2-approximation of the MRCT.

By Lemma 59.1, we can design a 2-approximation algorithm which constructs a shortest-paths tree rooted at each vertex and chooses the best of them. Since there are only n vertices and a shortest-paths tree can be constructed in $O(n \log n + m)$ time, the algorithm runs in $O(n^2 \log n + mn)$ time, which is the same as the result stated in Corollary 59.1.

59.2.2 Routing Loads, Separators, and General Stars

We introduce a term, *routing load*, which provides us an alternative formula to compute the routing cost of a tree. For any edge $e \in E(T)$, let x and y, $x \leq y$, be the number of vertices in the two subtrees resulting by removing e. The routing load on e, denoted by $l(T, e)$, is $2xy = 2x(n - x)$. Notice that $x \leq n/2$, and the routing load increases as x increases. The following property can be easily shown by definition:

Fact 59.1

For any edge $e \in E(T)$, if the number of vertices in both sides of e are at least δn, the routing load on e is at least $2\delta(1 - \delta)n^2$. Furthermore, for any edge of a tree T, the routing load is upper-bounded by $n^2/2$.

Intuitively the routing load is the number of paths passing through the edge. To compute the routing load of an edge of a tree, all we need to do is to compute the number of vertices in both sides of the edge. By rooting the tree at an arbitrary vertex and traveling in a post order, we can compute the routing load of every edge in linear time.

Lemma 59.2

For a tree T with edge length w, $c(T) = \sum_{e \in E(T)} l(T, e)w(e)$. In addition, $c(T)$ can be computed in $O(n)$ time.

A key point to the 2-approximation in the last section is the existence of the centroid, which separates a tree into sufficiently small components. To generalize the idea, we define the separator as follows. Let $\delta \leq 1/2$. Root a tree T at its centroid, and then remove all the vertices of which the number of descendants (including itself) are equal to or less than δn. The remaining subgraph is defined as a *minimal δ-separator* of T. Obviously, the separator is a connected subgraph. A star is a tree with only one internal vertex (center). We define a *general star* as follows:

Definition 59.1

Let R be a tree contained in graph G. A spanning tree T is a general star with core R if each vertex is connected to R by a shortest path.

Let S be a connected subgraph of a spanning tree T. Let $d_T(v, S)$ denote the minimum distance from v to any vertex of S on the graph T. For a graph G and $u, v \in V(G)$, we use $SP_G(u, v)$ to denote a shortest path between u and v in G. For convenience, we define $d_T^S(u, v) = w(SP_T(u, v) \cap S)$. Obviously $d_T(u, v) \leq d_T(v, S) + d_T^S(u, v) + d_T(u, S)$, and the equality holds if v and u are in different branches. Summing up the inequality for all pairs of vertices, we have

$$c(T) \leq 2n \sum_{v \in V} d_T(v, S) + \sum_{u,v \in V} d_T^S(u, v)$$

By the definition of routing load,

$$\sum_{u,v \in V} d_T^S(u, v) = \sum_{e \in E(S)} l(T, e)w(e)$$

Suppose that T is a general star with core S. We can establish an upper bound of the routing cost by observing that $d_T(v, S) = d_G(v, S)$ for any vertex v and $l(T, e) \leq \frac{n^2}{2}$ for any edge e (Fact 59.1).

Lemma 59.3

If T is a general star with core S, $c(T) \leq 2n \sum_{v \in V(G)} d_G(v, S) + (n^2/2)w(S)$.

Let S be a minimal δ-separator of a spanning tree T. The following lower bound of the minimum routing cost was established:

Lemma 59.4

If S is a minimal δ-separator of $\widehat{T}$, then

$$c(\widehat{T}) \geq 2(1 - \delta)n \sum_{v \in V} d_{\widehat{T}}(v, S) + 2\delta(1 - \delta)n^2 w(S)$$

59.2.3 Approximating by a General Star

As shown previously, a 1/2-separator is used to derive a 2-approximation algorithm. The idea is now generalized to show that a better approximation ratio can be obtained by using a 1/3-separator. Let r be a centroid of T. There are at most two branches of r with more than $n/3$ vertices. Therefore, there exists a path $P \subset T$ such that P is a 1/3-separator of T, and we say that P is a *path separator* of T.

Substituting $\delta = 1/3$ in Lemma 59.4, we obtain a lower bound of the minimum routing cost.

Corollary 59.2

If P is a path separator of $\widehat{T}$, then

$$c(\widehat{T}) \geq \frac{4n}{3} \sum_{v \in V} d_{\widehat{T}}(v, P) + \frac{4n^2}{9} w(P)$$

The following result can then be shown by Lemma 59.3 and Corollary 59.2:

Lemma 59.5

There exists $r_1, r_2 \in V$ such that if $R = SP_G(r_1, r_2)$ and T is a general star with core R, then $c(T) \leq (15/8)c(\widehat{T})$.

By Lemma 59.5 we have a 15/8-approximation algorithm for the MRCT problem. For every r_1 and r_2 in V, we construct a shortest path $R = SP_G(r_1, r_2)$ and a general star T with core R. The one with the minimum routing cost must be a 15/8-approximation of the MRCT. All-pairs shortest paths can be found in $O(n^3)$ time. A direct method takes $O(n \log n + m)$ time for each pair r_1 and r_2, and therefore $O(n^3 \log n + n^2 m)$ time in total. By avoiding some redundant computations, the time complexity can be reduced to $O(n^3)$, and the following result was obtained:

Theorem 59.2

There is a 15/8-approximation algorithm for the MRCT problem with time complexity $O(n^3)$.

Let P be a path separator of an optimal tree. By Lemma 59.3, if X is a general star with core P, then

$$c(X) \leq 2n \sum_{v \in V} d_G(v, P) + (n^2/2)w(P)$$

By Lemma 59.2, it can be shown that X is a 3/2-approximation solution. However, it costs exponential time to try all possible paths. Let $P = (p_1, p_2, \ldots, p_k)$. It is easy to see that a centroid must be in $V(P)$. Let p_q be a centroid of $\widehat{T}$. Construct $R = SP_G(p_1, p_q) \cup SP_G(p_q, p_k)$. Let T be any general star with core R. One can show that such T is a 3/2-approximation. For every triple (r_1, r_0, r_2) of vertices, we construct $R = SP_G(r_1, r_0) \cup SP_G(r_0, r_2)$ and find a general star with core R. The one with the minimum routing cost is a 3/2-approximation.

Theorem 59.3

The MRCT can be approximated with error ratio 3/2 in $O(n^4)$ time.

59.2.4 A Reduction to the Metric Case

Let S be a minimal δ-separator of $\widehat{T}$. The strategy of algorithms shown above is to "guess" the structure of S and to construct a general star with the guessed structure as the core. If T is a general star with core S, by Lemmas 59.3 and 59.4,

$$c(T) \leq 2n \sum_{v \in V(G)} d_G(v, S) + (n^2/2)w(S)$$

and

$$c(\widehat{T}) \geq 2(1-\delta)n \sum_{v \in V} d_{\widehat{T}}(v, S) + 2\delta(1-\delta)n^2 w(S)$$

The approximation ratio, by comparing the two inequalities, is $\max\{\frac{1}{1-\delta}, \frac{1}{4\delta(1-\delta)}\}$. The ratio achieves its minimum when the two terms coincide, that is, $\delta = 1/4$, and the minimum ratio is 4/3. In fact, by using a general star and a (1/4)-separator, it is possible to approximate an MRCT with ratio $(4/3) + \varepsilon$ for any constant $\varepsilon > 0$ in polynomial time. The additional error ε is due to the difference between the guessed and the true separators.

By this strategy, the approximation ratio is limited even if S was known exactly. The limit of the approximation ratio may be mostly due to that we consider only general stars. In a general star, the vertices are always connected to their closest vertices of the core. In extreme cases, there are roughly half of the vertices connected to both sides of a costly edge. This results in the cost $(n^2/2)w(S)$ in the upper bound of a general star. To make a breakthrough, the restriction that each vertex must be connected to the closest vertex of the core needs to be relaxed.

A metric graph is a complete graph with triangle inequality, that is, each edge is a shortest path of its two endpoints. Define k-stars to be the trees with at most k internal vertices. Importantly, k-stars have no such restriction like general stars and can be used to approximate an MRCT more precisely. However, k-stars work only for metric graphs. So we should first clarify the computational complexity and the approximability of the MRCT problem on metric graphs. The following transformation provides us the answers.

The metric closure of a graph $G = (V, E, w)$ is the complete graph $\bar{G} = (V, V \times V, \bar{w})$ in which $\bar{w}(u, v) = d_G(u, v)$ for all $u, v \in V$. Any edge (a, b) in $\bar{G}$ is called a *bad edge* if $(a, b) \notin E$ or $w(a, b) > \bar{w}(a, b)$. It was shown that given any spanning tree T of $\bar{G}$, in $O(n^3)$ time, we can construct another spanning tree Y without any bad edge such that $c(Y) \leq c(T)$. Since Y has no bad edge, it can be thought of as a spanning tree of G with the same routing cost. As a result, we have the next theorem, in which ΔMRCT denotes the MRCT problem with metric inputs. By the transformation, it is straightforward that the ΔMRCT problem is NP-hard.

Theorem 59.4

If there is an approximation algorithm for ΔMRCT with time complexity $O(f(n))$, then there is an approximation algorithm for MRCT with the same approximation ratio and time complexity $O(f(n) + n^3)$.

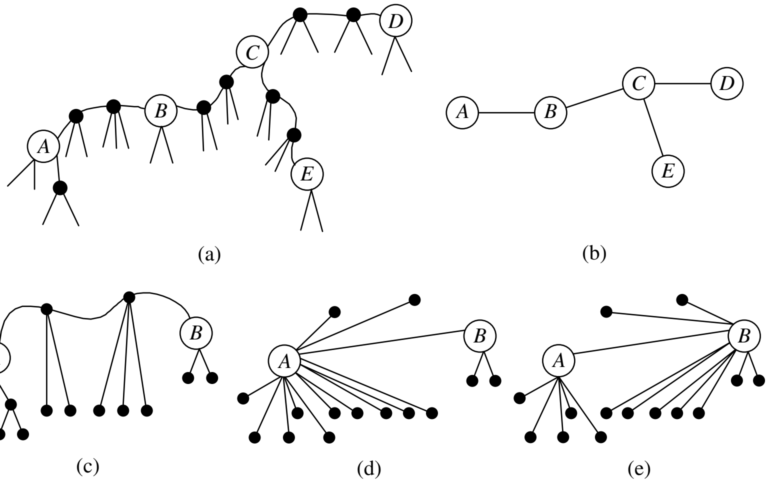
59.2.5 A Polynomial-Time Approximation Scheme

By Theorem 59.4, we may focus on metric graphs. The k-stars, that is, trees with no more than k internal nodes, are used as a basis of the approximation scheme. The design of the PTAS consists of two parts: the existence of a k-star which is a $(k+3)/(k+1)$-approximation and how to compute the best k-star in polynomial time.

59.2.5.1 Approximation Ratio

Root $\widehat{T}$ at its centroid r. For a desired positive $\delta \leq 1/2$, removing all vertices with no more than δn descendants, we obtain a minimal δ-separator S. We then choose some critical vertices, defined as the *cut and leaf set*, to partition S into some edge-disjoint paths, called as δ-paths, each of which has only few nodes (at most $\delta n/2$) hanging at its internal nodes. It was shown that the number of the necessary critical vertices is at most $2/\delta - 3$. By the following steps, we construct a k-star used to argue the upper bound on the routing cost:

1. Replace the δ-paths with the short-cutting edges to construct a tree structure.
2. All vertices in subtrees hanging at the cut and leaf nodes are connected directly to their closest node in the tree.



(a) (b)

(c) (d) (e)

FIGURE 59.2 Constructing the k-star from an optimal tree.

3. Along a δ-path, all the internal nodes and nodes in subtrees hanging at internal nodes are connected to one of the two endpoints of this path (notice that both are in the cut and leaf set) in such a way as to minimize the resulting routing cost.

In Figure 59.2, we illustrate how to construct the desired k-star from an optimal tree. Frame (a) is an optimal tree in which the separator is shown and the cut and leaf set is $\{A, B, C, D, E\}$. Frame (b) is the tree spanning the cut and leaf nodes, which has the same skeletal structure as the separator. Frames (c)–(e) illustrate how to connect other nodes to the cut and leaf nodes. Frame (c) exhibits the nodes hanging at a δ-path. These nodes will be connected as in either Frame (d) or (e). The nodes hanging at the endpoints of the path will be connected to the endpoints in either case. All the internal nodes of the path and nodes hanging at the internal nodes will be connected to one of the two endpoints. Notice that they are connected to the same endpoint either as Frame (d) or Frame (e), but not connected to the two endpoints partially.

By establishing a more precise lower bound than Lemma 59.4 and using the properties of δ-separator and δ-paths, it can be shown that the routing cost of such a k-star is at most $1/(1-\delta)$ times the optimal. For any integer $k \geq 1$, we take $\delta = \frac{2}{k+3}$, and obtain the next result.

Lemma 59.6

A k-star of minimum routing cost is a $(k+3)/(k+1)$-approximation of an MRCT.

59.2.5.2 Finding the Optimal k-Star

For a given k, to find an optimal k-star, we consider all possible subsets S of vertices of size k, and for each such choice, find an optimal k-star where the remaining vertices have degree one. Any k-star can be described by a triple $(S, \tau, \mathcal{L})$, where $S = \{v_1, \ldots, v_k\} \subseteq V$ is the set of k distinguished vertices which may have degree more than one, τ a spanning tree topology on S, and $\mathcal{L} = (L_1, \ldots, L_k)$, where $L_i \subseteq V - S$ is the set of vertices connected to vertex $v_i \in S$. For any $r \in Z^+$, an r-vector is an integer vector with r components. Let $l = (l_1, \ldots, l_k)$ be a nonnegative k-vector such that $\sum_{i=1}^{k} l_i = n - k$. We say that a k-star $(S, \tau, \mathcal{L})$ has the configuration (S, τ, l) if $l_i = |L_i|$ for all $1 \leq i \leq k$.

For a fixed k, the total number of configurations is $O(n^{2k-1})$ since there are $\binom{n}{k}$ choices for S, k^{k-2} possible tree topologies on k-vertices, and $\binom{n-1}{k-1}$ possible such k-vectors. Notice that any two k-stars with the same configuration have the same routing load on their corresponding edges. Any edge cross the cut $(S, V-S)$, connects a leaf to a node in S, and therefore has the same routing load $2(n-1)$. Since all these routing loads are the same, the best way of connecting the vertices in $V - S$ to nodes in S can be solved in polynomial time for a given configuration by a straightforward reduction to an instance of minimum-cost perfect matching. The minimum-cost perfect matching problem, also called the *assignment* problem, has been well studied and can be solved in $O(n^3)$ time [2]. Therefore, the overall complexity is $O(n^{2k+2})$ for finding an optimal k-star.

In the PTAS, we need to solve many matching problems, each for one configuration. It takes polynomial time to solve these problems individually and this result is sufficient for showing the existence of the PTAS. Although there is no obvious way to reduce the time complexity for one matching problem, the total time complexity can be significantly reduced when considering all these matching problems together. By carefully ordering the matching problems for the configurations and exploiting the common structure of two consecutive problems, it was shown that the best k-star for any configuration in this order can be obtained from the optimal solution of the previous configuration in $O(nk)$ time. As a result, an optimal k-star of a metric graph can be constructed in $O(n^{2k})$ time. The next theorem concludes the result for the PTAS of the MRCT.

Theorem 59.5

There is a PTAS for finding a minimum routing cost tree of a weighted undirected graph. Specifically, we can find a $(1+\varepsilon)$-approximation solution in time $O(n^{2\lceil \frac{2}{\varepsilon}\rceil - 2})$.

59.3 Product-Requirement Communication Spanning Tree

The *product-requirement communication* (or p.r.c. in abbreviation) cost of a tree T is defined by $c_p(T) = \sum_{u,v} r(u)r(v)d_T(u, v)$, in which r is a nonnegative vertex weight. Recall that the PTAS for the MRCT problem is obtained by showing the following properties:

1. The MRCT problem on general graphs is equivalent to the problem on metric graphs.
2. The best k-star is a $((k+3)/(k+1))$-approximation solution for the metric MRCT problem.
3. For a fixed k, the best k-star of a metric graph can be found in polynomial time.

The PROCT problem is a weighted counterpart of the MRCT problem. A vertex with weight $r(v)$ can be regarded as a super node consisting of $r(v)$ nodes of unit weight and connected by edges of zero length. In fact, the first and the second properties remain true for the PROCT problem. They can be obtained by straightforward generalizations of the previous results. However, there is no obvious way to generalize the algorithm for the minimum routing cost k-star to that for the minimum p.r.c. cost k-star. A straightforward generalization conducts to a pseudopolynomial-time algorithm whose time complexity depends on the total weight of all vertices.

For convenience, we define a *balanced k-star* by adding a restriction that the core must be a minimal $(2/(k+3))$-separator of the spanning tree. The performance ratio of an optimal balanced k-star is the same as in Lemma 59.6. Therefore, to approximate a PROCT, we can only focus on the problem of finding an optimal balanced k-star on a metric graph. When k is a constant, the number of all possible cores is polynomial, and we may reduce the problem to the following subproblem:

PROBLEM: Optimal Balanced k-Stars with a Given Core
INSTANCE: A metric graph $G = (V, E, w)$, a tree S in G with $|V(S)| = k$, and a vertex weight $r : V \to Z^+$.
GOAL: Find an optimal balanced k-star with core S if it exists.

In this section, two approximation algorithms are presented. The first algorithm approximates a PROCT by finding the minimum p.r.c. cost 2-star, and the second one is a PTAS, which employs a PTAS for a minimum p.r.c. cost k-star.

For a vertex set U, we use $r(U)$ to denote $\sum_{u \in U} r(u)$, and $r(H) = r(V(H))$ for a graph H. Let $R = r(G)$ denote the total vertex weight of the input graph. Similar to a centroid of an unweighted graph, we define the r-centroid of a tree with vertex weight function r as follows. Let T be a tree with vertex weight function r. The *r-centroid* of a tree T is a vertex $m \in V(T)$ such that if we remove m, then $r(H) \leq r(T)/2$ for any branch H. The p.r.c. routing load on the edge e of a tree T is defined by $l_p(T, e) = 2r(T_u)r(T_v)$, where T_u and T_v are the two subgraphs obtained by removing e from T. The p.r.c. routing cost on the edge e is defined to be $l_p(T, e)w(e)$. Similarly, the p.r.c. routing cost of a tree can also be computed by summing up the edge lengths multiplied by their p.r.c. routing load.

Lemma 59.7

Let T be any spanning tree of a graph $G = (V, E, w)$ and r be a vertex weight function. $c_p(T) = \sum_{e\in E(T)} l_p(T, e)w(e)$.

59.3.1 Approximating by 2-Stars

The core of a 2-star is an edge and there are $O(n^2)$ possible cores. A 2-star T can be represented by an edge (x, y) and a bipartition (X, Y) of V, in which X and Y are the sets of nodes connected to x and y, respectively. By definition, its cost can be calculated by the following formula:

$$c_p(T) = 2r(X)r(Y)w(x, y) + 2\sum_{v\in X} r(v)(R - r(v))w(x, v) + 2\sum_{v\in Y} r(v)(R - r(v))w(y, v) \quad (59.1)$$

For each possible edge (x, y), the goal is to find the corresponding bipartition such that the p.r.c. routing cost is minimized. It is not hard to find that such a bipartition can be found by solving a minimum-cut problem on an auxiliary graph. Since the minimum cut of a graph can be found in $O(n^3)$ [3], the minimum p.r.c. cost 2-star can be found in $O(n^5)$ time. By a result similar to Lemma 59.6, such a 2-star is a (5/3)-approximation of a PROCT. In the following, with a more precise analysis, we show that the approximation ratio is 1.577.

Let T be a PROCT of the metric graph. We are going to construct two 2-stars and show that one of them is a 1.577-approximation of T. First we establish a lower bound of the optimal cost. Let $1/3 < \delta < 0.5$ be a real number to be determined later. Since $\delta > 1/3$, there exists a path P, which is a minimal δ-separator of T. Let a and b be the two endpoints of P. Similar to Lemma 59.4, we have

$$c_p(T) \geq 2\,(1 - \delta)\,R\sum_{x} r(x)d_T(x, P) + 2\delta(1 - \delta)R^2w(P) \quad (59.2)$$

Let V_a and V_b be the sets of vertices which are connected to P at a and b, respectively. Consider two 2-stars T^* and T^{**} with the same core (a, b) and their corresponding bipartitions are $(V - V_b, V_b)$ and $(V_a, V - V_a)$, respectively. By Eq. (59.1) and the triangle inequality, we can show that

$$\min\{c_p(T^*), c_p(T^{**})\} \leq 2R\sum_{v\in V} r(v)d_T(v, P) + (1 - 2\delta^2)R^2w(P) \quad (59.3)$$

By Eq. (59.2) and Eq. (59.3), the approximation ratio is $\max\{1/(1-\delta), (1-2\delta^2)/(2\delta(1-\delta))\}$, in which $1/3 < \delta < 1/2$. By setting $\delta = (\sqrt{3} - 1)/2 \simeq 0.366$, we get the ratio 1.577. Combining with the time complexity of finding the minimum p.r.c. cost 2-star and the reduction from general to metric graphs, we obtain the next result.

Theorem 59.6

A PROCT of a general graph can be approximated with ratio 1.577 in $O(n^5)$ time.

59.3.2 A Polynomial-Time Approximation Scheme

We now show that the PROCT problem admits a PTAS. Instead of finding a minimum p.r.c. cost k-star exactly, the PTAS finds an approximation of an optimal balanced k-star. Let $G = (V, E, w)$ be a metric graph and S a given core consisting of k vertices. Let $U = V - V(S)$. The goal is to connect every vertex in U to the core so as to make the p.r.c. cost as small as possible. For each vertex $v \in U$, we regard v as a super node consisting of $r(v)$ nodes of weight one and connected by zero-length edges. Since all these nodes have weight one, by the technique in the PTAS of MRCT, the best leaf connection can be found by solving a series of assignment problems. The time complexity is $O(R^k)$, in which $R = r(V)$ is the total vertex weight. However, the time complexity depends on the total weight of the vertices, that is, it is a pseudopolynomial-time algorithm.

To reduce the time complexity, a natural idea is to scale down the weight of each vertex by a common factor. Since the algorithm works only on vertices of integer weights, there are rounding errors. To ensure the quality of the solution, some details of the algorithm should be designed carefully and we need to show that the rounding errors on the vertex weights do not affect too much the cost of the solution.

By a selected threshold, we first divide U into a light part and a heavy part according to their weights. Then, for each vertex in the heavy part, we scale down their weights by a scaling factor and round them to integers. When the weights are all integers, the best connection (with respect to the scaled weights) can be determined by a pseudopolynomial-time algorithm. Finally, each vertex in the light part is connected to its closest vertex in S. It will be shown that the approximation ratio and time complexity are determined by k, the scaling factor, and the threshold for dividing the vertices into the light and heavy parts. The PTAS is given below.

```
Algorithm PTAS_STAR
Input: A metric graph $G = (V, E, w)$ with vertex weight $r$, a tree $S$ in $G$
    with $|V(S)| = k$, a positive number $\lambda < 1$ and a positive integer $q$.
Output: A $k$-star with core $S$.
    /* assume that $V(S) = \{s_i | 1 \leq i \leq k\}$ and $U = \{1..n - k\}$
    in which $r(i) \leq r(i + 1)$ for each $i \in U$. */
1: Find the maximum $j$ such that $r(\{1..j\}) \leq \lambda R$.
    Let $V_L = \{1..j\}$ and $V_H = \{j + 1..n - k\}$ and $\mu = r(j + 1)$.
2: Let $\bar{r}(v) = \lfloor qr(v)/\mu \rfloor$ for each $v \in V_H$;
    and $\bar{r}(v) = qr(v)/\mu$ for each $v \in V(S)$.
3: Find an optimal $k$-star $T_1$ with respective to $\bar{r}$.
4: Construct $T$ from $T_1$ by connecting each vertex in $V_L$
    to the closest vertex in $S$.
5: Output $T$.
```

The time complexity and approximation ratio are shown in the next theorem. The approximation ratio approaches to 1 as q and λ^{-1} go to infinity. Therefore, for any desired approximation ratio $1 + \varepsilon > 1$, we can choose suitable q and λ, and the time complexity is polynomial when they are fixed.

Theorem 59.7

Algorithm PTAS_STAR *is a PTAS for an optimal balanced k-star with a given core. For any positive integer q and positive number* $\lambda < 1$, *the time complexity is* $O((nq/\lambda)^k)$ *and approximation ratio is* $((1 + q^{-1})^2 + \lambda(k + 3)^2/(k + 1))$.

We conclude the section by the next theorem.

Theorem 59.8

The PROCT problem on general graphs admits a PTAS.

59.4 Sum-Requirement Communication Spanning Tree

The *sum-requirement communication* (or s.r.c. in abbreviation) cost of a tree T is defined by $c_s(T) = \sum_{u,v}(r(u) + r(v))d_T(u, v)$. Similar to the PROCT problem, the SROCT problem includes the MRCT problem as a special case and is therefore NP-hard. The s.r.c. cost of a tree can also be computed by summing the routing costs of edges. The only difference is the definition of routing load. We define the s.r.c. routing load on the edge e to be $l_s(T, e) = 2(r(T_u)|V(T_v)| + r(T_v)|V(T_u)|)$, where T_u and T_v are the two subgraphs obtained by removing e from T.

Lemma 59.8

Let T be any spanning tree of a graph $G = (V, E, w)$ and r be a vertex weight function. $c_s(T) = \sum_{e\in E(T)} l_s(T, e)w(e)$.

For the PROCT problem, it has been shown that an optimal solution for a graph has the same value as the one for its metric closure. In other words, using bad edges cannot lead to a better solution. However, the SROCT problem has no such property. By giving a counterexample, it was shown that a bad edge may reduce the s.r.c. cost. It is still unknown if any approximation algorithm on metric graphs ensures the same approximation ratio for general graphs.

In this section, we introduce a 2-approximation algorithm for the SROCT problem on general graphs. For each vertex v, the algorithm finds the shortest-paths tree rooted at v. Then it outputs the shortest-paths tree with minimum s.r.c. cost. Obviously, the time complexity of the algorithm is $O(n^2 \log n + mn)$ since it constructs $O(n)$ shortest-paths trees and each takes $O(n \log n + m)$ time. Similar to the MRCT problem, we obtain the approximation ratio by showing that there always exists a vertex x such that any shortest-paths tree rooted at x is a 2-approximation solution.

We use $\widehat{T}$ to denote an optimal spanning tree of the SROCT problem, and use x_1 and x_2 to denote a centroid and an r-centroid of $\widehat{T}$, respectively. Let P be the path between the two vertices x_1 and x_2 on the tree. For any edge $e \in E(P)$, let T_1 and T_2 be the two subtrees resulting by deleting e from $\widehat{T}$. Assume that $x_1 \in V(T_1)$ and $x_2 \in V(T_2)$. By definition, $|V(T_1)| \geq n/2$ and $r(T_2) \geq R/2$, in which $R = r(V)$. Then,

$$l_s(\widehat{T}, e)/2 = |V(T_1)|r(T_2) + |V(T_2)|r(T_1) = 2\,(|V(T_1)| - n/2)\,(r(T_2) - R/2) + nR/2 \geq nR/2$$

By the inequality, we are able to establish a lower bound of the minimum s.r.c. cost. Recall that $d_{\widehat{T}}(v, P)$ denotes the shortest-path length from vertex v to path P.

Lemma 59.9

$c_s(\widehat{T}) \geq \sum_{v\in V} (nr(v) + R)\, d_{\widehat{T}}(v, P) + nRw(P)$.

For any vertex v, let $f_1(v) = d_{\widehat{T}}(v, x_1) - d_{\widehat{T}}(v, P)$ and $f_2(v) = d_{\widehat{T}}(v, x_2) - d_{\widehat{T}}(v, P)$. It should be noted that $f_1(v) + f_2(v) = w(P)$. Let Y^* and Y^{**} be the shortest-paths trees rooted at x_1 and x_2, respectively. By the triangle inequality, we can obtain that $c_s(Y^*) \leq 2\sum_{v\in V} (nr(v) + R)\,(d_{\widehat{T}}(v, P) + f_1(v))$ and $c_s(Y^{**}) \leq 2\sum_{v\in V} (nr(v) + R)\,(d_{\widehat{T}}(v, P) + f_2(v))$. Taking the mean of the two inequalities, we have

$$\min\{c_s(Y^*), c_s(Y^{**})\} \leq 2\sum_{v\in V} (nr(v) + R)\, d_{\widehat{T}}(v, P) + 2nRw(P)$$

which is at most $2c_s(\widehat{T})$ by Lemma 59.9. The next theorem summarizes the result in this section.

Theorem 59.9

There exists a 2-approximation algorithm with time complexity $O(n^2 \log n + mn)$ for the SROCT problem.

59.5 Multiple Sources MRCT

In the multiple sources MRCT (p-MRCT) problem, we are given a set $S \subset V$ of p sources, and the cost function is defined by $c_m(T) = \sum_{u\in S} \sum_{v\in V} d_T(u, v)$. If there is only one source, the problem is reduced to the shortest-paths tree problem, and therefore the 1-MRCT problem is polynomial-time solvable. For the other extreme case that all vertices are sources, the problem is reduced to the MRCT problem, and is therefore NP-hard. By a transformation from the well-known *satisfiability problem* [4,5], it was shown that the p-MRCT problem is also NP-hard for any fixed $p > 1$ even when the input is a metric graph [15].

Recall that in the SROCT problem, the objective function is $c_s(T) = \sum_{u,v}(r(u) + r(v))d_T(u, v)$ in which r is the given vertex weight. By setting $r(v) = 1$ for each $v \in S$ and $r(v) = 0$ for other vertices, it is easy to see that the p-MRCT problem is just a special case of the SROCT problem, and therefore can be approximated with ratio 2.

Theorem 59.10

The p-MRCT problem admits a 2-approximation algorithm with time complexity $O(n^2 \log n + mn)$.

59.5.1 Approximating the 2-MRCT

For the 2-MRCT problem, there are only two sources. We shall assume that s_1 and s_2 are the given sources. Let T be a tree and P the path between the two sources in T. For any $v \in V$, $d_T(v, s_1) + d_T(v, s_2) = w(P) + 2d_T(v, P)$. Summing over all vertices in V, we obtain that $c_m(T) = nw(P) + 2\sum_{v\in V} d_T(v, P)$. Therefore, once a path P between the two sources has been chosen, it is obvious that the best way to extend P to a spanning tree is to add the shortest-paths forest using the vertices of P as multiple roots, that is, the distance from each vertex to the path is made as small as possible. To approximate the 2-MRCT, it is natural to connect the two sources by a shortest path.

Algorithm 2MRCT

```
1: Find a shortest path $P$ between $s_1$ and $s_2$ on $G$.
2: Find the shortest-paths forest with multiple roots in $V(P)$.
3: Output the tree $T$ which is the union of the forest and $P$.
```

We are going to show the performance of the algorithm. First we establish a lower bound of the optimum. Let $\widehat{T}$ be an optimal tree of the 2-MRCT problem. Since $d_{\widehat{T}}(v, s_i) \geq d_G(v, s_i)$ for any vertex v and for $i \in \{1, 2\}$, the optimal cost is lower-bounded by $\sum_{v\in V}(d_G(v, s_1) + d_G(v, s_2))$. By the triangle inequality, we may obtain another lower bound $nd_G(s_1, s_2)$. By taking the mean of the two lower bounds, we have

$$c_m(\widehat{T}) \geq \frac{1}{2}\sum_{v}(d_G(v, s_1) + d_G(v, s_2)) + \frac{n}{2}d_G(s_1, s_2) \qquad (59.4)$$

Let T be the tree constructed by Algorithm 2MRCT and P a shortest path between the two sources. Since each vertex is connected to P by a shortest path, for any vertex v,

$$d_T(v, P) \leq \min\{d_G(v, s_1), d_G(v, s_2)\} \leq \frac{1}{2}(d_G(v, s_1) + d_G(v, s_2))$$

Therefore $c_m(T) \leq nd_G(s_1, s_2) + \sum_v(d_G(v, s_1) + d_G(v, s_2))$. Comparing with Eq. (59.4), we have $c_m(T) \leq 2c_m(\widehat{T})$. Since the total time complexity is dominated by the step of finding the shortest-paths tree, we have the next result.

Theorem 59.11

The 2MRCT algorithm finds a 2-approximation of a 2-MRCT in $O(n \log n + m)$ time.

The ratio shown in Theorem 59.11 is tight in the sense that there exists an instance such that the spanning tree constructed by the algorithm has a routing cost twice as the optimum. Consider a complete graph in which $w(v, s_1) = w(v, s_2) = 1$ and $w(s_1, s_2) = 2$ for each vertex v. The distance between any other pair of vertices is zero. At Step 1, the algorithm may find edge (s_1, s_2) as the path P, and then all other vertices are connected to one of the two sources. The routing cost of the constructed tree is $4n - 4$. On an optimal tree, the path between the two sources is a two-edge path, and all other vertices are connected to the middle vertex of the path. The optimal routing cost is therefore $2n$. The increased cost is due to missing the vertex on the path. In contrast, the existence of the vertex reduces the cost at an amount of $w(P)$ for each vertex.

The worst-case instance of the simple algorithm gives us some intuitions to improve the error ratio. To reduce the error, we may try to guess some vertices of the path. Let r be a vertex of the path between the two sources on an optimal tree and U the set of vertices connected to the path at r. If the path P found in Step 1 of the simple algorithm includes r, the distance from any vertex in U to each of the sources will be no more than the corresponding distance on the optimal tree. In addition, the vertex r partitions the path into two subpaths. The maximal increased cost by one of the vertices is the length of the subpath instead of the whole path. In the following, we introduce a PTAS based on this idea. For the sake of convenience, we shall first assume that G is a metric graph, and the generalization to general graphs will be discussed later.

By at most k vertices on the path, P can be partitioned into $k+1$ subpaths such that the number of vertices hanging at the internal nodes of each subpath is at most $n/(k+1)$. Here a path or a subpath may consist of only one vertex. Suppose that $(P_0, P_1, \ldots, P_k)$ is such a partition of P and the endpoints of P_i are m_i and m_{i+1} for each $0 \leq i \leq k$, in which $m_0 = s_1$ and $m_{k+1} = s_2$. Let U_i be the set of vertices hanging at the internal nodes of P_i in $\widehat{T}$ and $U = V - \bigcup_{0 \leq i \leq k} U_i$ the set of the remaining vertices. Construct a path X from P by replacing each subpath P_i with the short-cut edge (m_i, m_{i+1}), and extend X to a spanning tree T by adding the shortest-paths forest using the vertices of X as multiple roots. By the construction of X, it is obvious that $w(X) \leq w(P)$. Similar to the proof of the 2-approximation, for any vertex $v \in U_i$, $0 \leq i \leq k$, the routing cost of v is increased by at most $w(P_i)$. Since $|U_i| \leq \frac{n}{k+1}$ and $\sum_i w(P_i) = w(P)$, the total increased cost is upper-bounded by $nw(P)/(k+1)$. Since $nw(P)$ is a lower bound of the optimal, T is a $((k+2)/(k+1))$-approximation of $\widehat{T}$.

Thus, once we correctly guess the k-vertices m_i, we can construct an approximation of the 2-MRCT with ratio $(k+2)/(k+1)$. By trying all possible k-tuples, we can ensure such an error ratio in $O(n^{k+1})$ time since there are $O(n^k)$ possible k-tuples and it takes $O(kn)$ time to connect the remaining vertices to their closest vertices in $V(X)$. For any $\varepsilon > 0$, we set $k = \lceil \frac{1}{\varepsilon} - 1 \rceil$, and the approximation ratio is $1 + \varepsilon$.

Theorem 59.12

The 2-MRCT problem on metric graphs admits a PTAS. For any constant $\varepsilon > 0$, a $(1+\varepsilon)$-approximation of a 2-MRCT of a graph G can be found in $O(n^{\lceil 1/\varepsilon \rceil})$ time.

To generalize the PTAS to general graphs, the only difficulty is how to construct a tree structure playing the same role as X in the above PTAS. The following result was shown to overcome the difficulty:

Lemma 59.10

Let $m_0, m_1, \ldots, m_{k+1}$ be k vertices in a general graph G and P be a path connecting the consecutive m_i. Given the graph G and m_i, in $O(kn^2)$ time, we can construct a tree X such that $m_i \in V(X)$ and $d_X(v, m_0) + d_X(v, m_{k+1}) \leq w(P)$ for any $v \in V(X)$.

As a result, the 2-MRCT problem on general graphs also admits a PTAS.

Theorem 59.13

The 2-MRCT problem on general graphs admits a PTAS. For any constant $\varepsilon > 0$, a $(1+\varepsilon)$-approximation of a 2-MRCT of a graph G can be found in $O(n^{\lceil 1/\varepsilon + 1 \rceil})$ time.

59.5.2 The Weighted 2-MRCT

We now turn to a weighted version of the 2-MRCT problem. In such a problem, we want to minimize $\sum_{v \in V}(\beta_1 d_T(s_1, v) + \beta_2 d_T(s_2, v))$, in which β_1 and β_2 are given positive real numbers. Without loss of generality, we define the objective function as $c_m(T, \beta) = \sum_{v \in V}(\beta d_T(s_1, v) + d_T(s_2, v))$, in which $\beta \geq 1$. As in Theorem 59.10, the weighted 2-MRCT admits a 2-approximation algorithm with time complexity $O(n^2 \log n + mn)$. We shall first present a more efficient 2-approximation algorithm, and then show that the problem admits a PTAS if the input is restricted to metric graphs.

59.5.2.1 On General Graphs

First we present the 2-approximation algorithm for general graphs. Basically each vertex is greedily connected to one of the two sources, and then the two sources are connected by a shortest path.

Algorithm W2MRCT

```
1: Partition $V$ into $(V_1, V_2)$ such that $v \in V_1$ if
       $(\beta + 1)d_G(v, s_1) + d_G(s_1, s_2) \le (\beta + 1)d_G(v, s_2) + \beta d_G(s_1, s_2)$;
2: For each $V_i$, with $s_i$ as the root, find a shortest-paths tree $T_i$ spanning $V_i$.
3: Find a shortest path between $s_1$ and $s_2$, and then
   connect $T_1$ and $T_2$ by inserting an edge of the path.
4: Output the tree obtained in the last step.
```

Let $\widehat{T}$ be the optimal tree and P the path from s_1 to s_2 on $\widehat{T}$. Let $f_1(v) = d_{\widehat{T}}(v, s_1) - d_{\widehat{T}}(v,P)$ and $f_2(v) = d_{\widehat{T}}(v, s_2) - d_{\widehat{T}}(v, P)$ for each vertex v. By the definition of routing cost, we have

$$c_m(\widehat{T}, \beta) = \sum_{v \in V}((\beta + 1)d_{\widehat{T}}(v, P) + \beta f_1(v) + f_2(v)) \tag{59.5}$$

Consider the cost in the case that vertex v is connected to s_1 by a shortest path. Since $d_G(v, s_1) \le d_{\widehat{T}}(v,P) + f_1(v)$ and $d_G(s_1, s_2) \le w(P) = f_1(v) + f_2(v)$, we have

$$\begin{aligned}(\beta + 1)d_G(v, s_1) + d_G(s_1, s_2) &\le (\beta + 1)d_{\widehat{T}}(v, P) + (\beta + 2) f_1(v) + f_2(v) \\ &= \beta d_{\widehat{T}}(v, s_1) + d_{\widehat{T}}(v, s_2) + 2 f_1(v)\end{aligned} \tag{59.6}$$

That is, the cost is increased by at most $2 f_1(v)$. Similarly, in the case that v is connected to s_2 by a shortest path, it can be shown that the cost is increased by at most $2\beta f_2(v)$. Consequently, the routing cost of v is increased by at most $\min\{2 f_1(v), 2\beta f_2(v)\}$. By taking a weighted mean of the two terms, we have

$$\begin{aligned}\min\{2 f_1(v), 2\beta f_2(v)\} &\le \frac{\beta^2}{\beta^2 + 1}(2 f_1(v)) + \frac{1}{\beta^2 + 1}(2\beta f_2(v)) \\ &= \frac{2\beta}{\beta^2 + 1}(\beta f_1(v) + f_2(v)) \\ &\le \beta f_1(v) + f_2(v)\end{aligned} \tag{59.7}$$

The last step is obtained by $2\beta \le \beta^2 + 1$ since $\beta^2 + 1 - 2\beta = (\beta - 1)^2 \ge 0$. Summing over all vertices and comparing with Eq. (59.5), we have that the approximation ratio is 2.

Theorem 59.14

For a general graph, a 2-approximation of the weighted 2-MRCT can be found in $O(n \log n + m)$ time.

59.5.2.2 On Metric Graphs

The main idea of the PTAS for the weighted case is similar to the unweighted one. We also try to guess k-vertices of the path between the two sources on the optimal tree. For each possible k-tuple $(m_1, m_2, \ldots, m_k)$ of vertices, we construct a path X starting at s_1, passing through the consecutive m_i, and ending at s_2. The only difference is the way to connect the remaining vertices to the path X. In the unweighted case, each remaining vertex is connected to the closest vertex in X. In the weighted case, the vertices are also connected to minimize the routing cost. But this time, due to the different cost function, we choose the vertex m_i such that $(\beta + 1)w(v, m_i) + \beta d_X(m_i, s_1) + d_X(m_i, s_2)$ is minimized.

By an analysis similar to the unweighted case, it can be shown that the constructed tree is a $(\frac{k+3}{k+1})$-approximation of the weighted 2-MRCT. Therefore it is a PTAS but is less efficient than one of the unweighted problems.

Theorem 59.15

The weighted 2-MRCT problem on metric graphs admits a PTAS. For any $\varepsilon > 0$, a $(1 + \varepsilon)$-approximation of the optimal can be found in $O(n^{\lceil 2/\varepsilon \rceil})$ time.

59.6 Multiple Sources OCT

In the p-OCT problem, the requirement between a source and a destination is any nonnegative number, whereas there is no requirement between two nonsource vertices. Let T be a tree and $S = \{s_1, s_2, \ldots, s_p\} \subset V(T)$ the set of given sources. For any vertex $v \in V(T)$, the communication cost of v on T is defined by $D_T(v) = \sum_{i=1}^{p} r_i(v) d_T(v, s_i)$, where $r_i(v)$ is the given nonnegative requirement between s_i and v. The communication cost of T is defined by $c(T) = \sum_{v \in V(T)} D_T(v)$.

59.6.1 The *p*-OCT on Metric Graphs

59.6.1.1 The 2-OCT

To approximate the 2-OCT, the algorithm starts at the edge (s_1, s_2), and then inserts other vertices one by one in an arbitrary order. In each iteration, we greedily connect a vertex v to either s_1 or s_2 depending on the communication cost. Precisely speaking, for each non-source vertex v, connect v to s_1 if

$$(r_1(v) + r_2(v))w(v, s_1) + r_2(v)w(s_1, s_2) \leq (r_1(v) + r_2(v))w(v, s_2) + r_1(v)w(s_1, s_2)$$

and connect to s_2 otherwise. We shall show that the approximation ratio is 2.

For the sake of convenience, we define some notations. Let $\widehat{T}$ be the 2-OCT and P the path between s_1 and s_2 on $\widehat{T}$. We define $f_1(v) = d_{\widehat{T}}(v, s_1) - d_{\widehat{T}}(v, P)$ and $f_2(v) = d_{\widehat{T}}(v, s_2) - d_{\widehat{T}}(v, P)$ for each vertex v. The next formula directly comes from the above notations and the definition of the communication cost.

$$c(\widehat{T}) = \sum_{v \in V} ((r_1(v) + r_2(v))d_{\widehat{T}}(v, P) + r_1(v) f_1(v) + r_2(v) f_2(v)) \tag{59.8}$$

Let T be the tree delivered by the greedy method. To show that T is a 2-approximation, it suffices to show that $D_T(v) \leq 2D_{\widehat{T}}(v)$ for any vertex v. By the triangle inequality, $w(v, s_1) \leq d_{\widehat{T}}(v, P) + f_1(v)$. By a similar analysis to Eq. (59.6), we can show that if v is connected to s_1, the cost of v is increased by at most $2 f_1(v) r_2(v)$, and the cost is increased by at most $2 f_2(v) r_1(v)$ if v is connected to s_2. Since the vertex v is connected to either s_1 or s_2 by choosing the minimum of the two costs, $D_T(v) \leq D_{\widehat{T}}(v) + \min\{2 f_1(v) r_2(v), 2 f_2(v) r_1(v)\}$.

Since the minimum of two number is no more than their weighted mean, we have

$$\begin{aligned} \min\{2 f_1(v) r_2(v), 2 f_2(v) r_1(v)\} &\leq \frac{r_1(v)^2}{r_1(v)^2 + r_2(v)^2} 2 f_1(v) r_2(v) + \frac{r_2(v)^2}{r_1(v)^2 + r_2(v)^2} 2 f_2(v) r_1(v) \\ &= \frac{2 r_1(v) r_2(v)}{r_1(v)^2 + r_2(v)^2} (f_1(v) r_1(v) + f_2(v) r_2(v)) \end{aligned} \tag{59.9}$$

Since $r_1(v)^2 + r_2(v)^2 - 2r_1(v)r_2(v) = (r_1(v) - r_2(v))^2 \geq 0$, we have

$$D_T(v) \leq D_{\widehat{T}}(v) + f_1(v) r_1(v) + f_2(v) r_2(v) \leq 2D_{\widehat{T}}(v) \tag{59.10}$$

Theorem 59.16

The greedy method finds a 2*-approximation of the* 2*-OCT of a metric graph.*

59.6.1.2 The *p*-OCT

To approximate the p-OCT, we define the *reduced skeleton* of a tree as follows. The S-skeleton of T is the subgraph obtained by repeatedly removing the nonsource leaves from T until all the leaves are sources. The reduced skeleton is obtained from the skeleton by eliminating the nonsource vertices of degree two. By eliminating a vertex of degree two, we mean that the two edges incident to the vertex is substituted by the short-cut edge. An example of the S-skeleton and reduced S-skeleton of a tree is illustrated in Figure 59.3.

The reduced skeleton is a tree spanning S and possibly some other vertices. By the definition and the property of a tree structure, it is not hard to show that the number of vertices of X is bounded by $2|S| - 2$. In other words, there are at most $|S| - 2$ nonsource vertices in X.

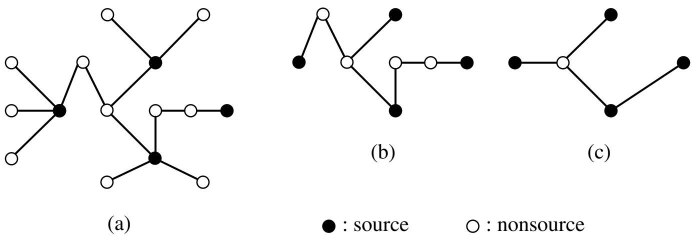


FIGURE 59.3 (a) A tree with four sources; (b) the skeleton; and (c) the reduced skeleton.

By introducing the reduced skeleton, the 2-approximation algorithm for the 2-OCT is generalized to the case of p sources, where $p \geq 2$ is a constant. The approximation algorithm tries to guess the reduced S-skeleton X of the OCT, and the other vertices are connected to one of the vertices of X by making the cost as small as possible. By a technique similar to the case of two sources, it was shown that the approximation ratio is 2.

The algorithm tries each tree X spanning the p sources and $(p-2)$ other vertices. The total number of such trees is $\binom{n-p}{p-2}(2p-2)^{2p-4}$. For each X and each $v \in V\backslash V(X)$, it takes $O(p)$ time to determine a vertex $u^* \in V(X)$ and insert edge (v, u^*). The total time complexity is therefore $O(n^{p-1})$ since p is a constant.

Theorem 59.17

For a metric graph, a 2*-approximation of the* p*-source OCT can be found in* $O(n^{p-1})$ *time, where* $p \geq 2$ *is a constant.*

59.6.2 The 2-OCT on General Graphs

In the following, we shall show that the 2MRCT algorithm in Section 59.5.1 is a 3-approximation algorithm of the 2-OCT problem in the case that the input is a general graph. Remember that the algorithm finds a shortest path between the two sources and then constructs a shortest-paths forest with all the vertices of the path as the multiple roots. The output tree is the union of the forest and the path. We now show the approximation ratio. Let $\widehat{T}$ be the 2-OCT and T the spanning tree obtained by the approximation algorithm. Suppose that v is connected to the path P at vertex x of P. In other words, among the trees of the shortest-paths forest, x is the root of the tree containing v. Therefore, $d_T(v, x) = d_G(v, x) \leq d_G(v, s_1)$. Since P is a shortest path between s_1 and s_2, $d_T(s_1, x) = d_G(s_1, x) \leq d_G(s_1, v) + d_G(v, x)$. Therefore,

$$d_T(v, s_1) = d_T(v, x) + d_T(x, s_1) \leq 2d_G(v, x) + d_G(s_1, v) \leq 3d_G(v, s_1)$$

Similarly, $d_T(v, s_2) \leq 3d_G(v, s_2)$. By the definition of the communication cost, it is easy to see that $D_T(v) \leq 3D_{\widehat{T}}(v)$. Since $c(T) = \sum_v D_T(v)$, we have that $c(T) \leq 3c(\widehat{T})$.

Theorem 59.18

The 2*MRCT algorithm finds a* 3*-approximation of the* 2*-OCT of a general graph in* $O(m + n \log n)$ *time.*

59.7 Concluding Remarks

The OCT problem was first discussed in Ref. [6], and the NP-hardness in the strong sense of the MRCT problem was shown in Ref. [7]. The first constant ratio approximation algorithm for the MRCT appeared in Ref. [1]. More details for the approximation algorithms surveyed in this chapter can be found in Ref. [8]. Besides approximation algorithms, exact algorithms for the MRCT have also been studied [9].

We close this chapter by mentioning a few open problems. First, it would be interesting to improve the approximation ratio for the weighted 2-MRCT of a general graph. By the previous result for the SROCT problem, the p-MRCT admits a 2-approximation algorithm for arbitrary p and for both weighted and unweighted cases. The 2-approximation algorithm in Section 59.5 only improves the time complexity. Although there is a PTAS for metric graphs, we did not find a similar result for general graphs.

Another open problem is how to approximate the p-OCT of general graphs. The $O(n^{p-1})$-time algorithm in Section 59.6 only works for metric graphs, and we did not find a similar result for general graphs. It would be nice to find a more efficient algorithm to approximate the p-OCT with good ratio. Another interesting issue concerns the development of an approximation scheme, by which one can control the trade-off between the time complexity and the approximation ratio.

References

[1] Wong, R., Worst-case analysis of network design problem heuristics. *SIAM J. Algebraic Discr.*, 1, 51, 1980.

[2] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., *Network Flows—Theory, Algorithms, and Applications*, Prentice-Hall, New York, 1993.

[3] Cormen, T. H., Leiserson, C. E., and Rivest, R. L., *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1994.

[4] Cook, S. A., The complexity of theorem-proving procedures, *Proc. of STOC*, 1971, p. 151.

[5] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, 1979.

[6] Hu, T. C., Optimum communication spanning trees, *SIAM J. Comput.*, 3, 188, 1974.

[7] Johnson, D. S., Lenstra, J. K., and Rinnooy Kan, A. H. G., The complexity of the network design problem, *Networks*, 8, 279, 1978.

[8] Wu B. Y. and Chao, K.-M., *Spanning Trees and Optimization Problems*, Chapman & Hall / CRC Press, USA, 2004.

[9] Fischetti, M., Lancia, G., and Serafini, P., Exact algorithms for minimum routing cost trees, *Networks*, 39, 161, 2002.

[10] Wu, B. Y., Chao, K.-M., and Tang, C. Y., Approximation algorithms for the shortest total path length spanning tree problem, *Discrete Appl. Math.*, 105, 273, 2000.

[11] Wu, B. Y., Lancia, G., Bafna, V., Chao, K.-M., Ravi, R., and Tang, C. Y., A polynomial time approximation scheme for minimum routing cost spanning trees, *SIAM J. Comput.*, 29, 761, 1999.

[12] Wu, B. Y., Chao, K.-M., and Tang, C. Y., Approximation algorithms for some optimum communication spanning tree problems, *Discrete Appl. Math.*, 102, 245, 2000.

[13] Wu, B. Y., Chao, K.-M., and Tang, C. Y., A polynomial time approximation scheme for optimal product-requirement communication spanning trees, *J. Algorithms*, 36, 182, 2000.

[14] Wu, B. Y., A polynomial time approximation scheme for the two-source minimum routing cost spanning trees, *J. Algorithms*, 44, 359, 2002.

[15] Wu, B. Y., Approximation algorithms for the optimal p-source communication spanning tree, *Discrete Appl. Math.*, 143, 31, 2004.

60

Approximation Algorithms for Multilevel Graph Partitioning

Burkhard Monien
University of Paderborn

Robert Preis
University of Paderborn

Stefan Schamberger
University of Paderborn

60.1 Introduction

The graph partitioning problem occurs in a wide range of applications. In its simplest form—the graph bisection problem—it is the task of dividing the vertices of a graph into two equally sized subsets such that the number of edges connecting vertices from both sets is minimal. Although easy to describe, this problem is known to be computationally difficult.

Due to the complexity of the problem, heuristics have to be applied to partition large graphs in a reasonable amount of time. A very powerful approach for this task is the multilevel graph partitioning paradigm [1–7]. The efficiency of this strategy depends mainly on two operations: graph coarsening and local improvement. For both subproblems methods with analytically proven worst-case performance are known.

For the coarsening part linear-time approximation algorithms computing a maximum-weighted matching in general edge-weighted graphs [8–11] exist. Although the maximum-weighted matching problem is solvable in polynomial time, it is not solvable in linear or close to linear time. Thus, these approximation algorithms are perfectly suitable in this context.

Most local improvement heuristics are based on the Kernighan–Lin (KL) algorithm [12]. However, this approach does not provide any valuable quality guarantee. An alternative approach is the Helpful-Set method [13,14], which origins from constructive proofs of upper bounds on the bisection width of regular graphs [7,15–17].

Overall, the combination of analytical methods for the two parts of the multilevel approach leads to an efficient graph-partitioning concept. A broader discussion can be found in, for example, Refs. [7,17].

In the remainder of this introduction we formally define the problem, give some bounds on the bisection width of graphs and discuss some applications. In Section 60.2 we refer to some standard global methods. In Section 60.3, we discuss applicable approximation algorithms for the multilevel graph partitioning paradigm. Finally, in Section 60.4, we point to some possible improvements.

60.1.1 Problem Definition

Given a graph $G = (V, E)$ with a set of vertices V and a set of edges E, the graph bisection problem can formally be defined the following way. Let $\pi : V \rightarrow \{1, 2\}$ be a partition (bisection) of the vertices from V into two sets V_1 and V_2. We define $\text{cut}(\pi) = |\{\{u, v\} \in E, \pi(u) \neq \pi(v)\}|$ to be the *cut-size* of π and $\text{bal}(\pi) = ||V_1| - |V_2||$ its balance. When solving the bisection problem, one looks for a partition π with $\text{bal}(\pi) \leq 1$ that minimizes $\text{cut}(\pi)$. This minimal cut-size is also called the *bisection width* of the graph. Refs. [18,19] show that the graph bisection problem is **NP**-complete.

The bisection problem can be generalized in several ways. First, weight functions $w_V : V \rightarrow \mathbb{R}$ on the vertices and $w_E : E \rightarrow \mathbb{R}$ on the edges can be introduced. Now, the sum of vertex weights in each part should be almost equal and the sum of weights of edges incident to vertices of different parts has to be minimized.

Furthermore, one could ask to divide the graph into more than two parts $V_1, \ldots, V_k$, which leads to the k-partitioning problem. In this case one looks for a partition $\pi : V \rightarrow \{1, \ldots, k\}$, which is defined to be *balanced* if $bal(\pi) := \max_{1 \leq i \leq k}\{\sum_{v \in V_i} w_V(v)\} - \frac{1}{k} \sum_{v \in V} w_V(v)$ is smaller than $\max_{v \in V} w_V(v)$. The cut-size of π is now defined as $\text{cut}(\pi) = \sum_{\{u,v\} \in E, \pi(u) \neq \pi(v)\}} w_E(\{u, v\})$.

Of course, once a bisection algorithm is present, it can be applied to partition a graph into more than two parts by recursive invocation, but in general the direct approach finds better solutions. A study of the resulting partition qualities of the direct and recursive approach is given, for example, in Refs. [20,21].

So far, we have discussed edge partitions (also known as edge separators). However, some applications require a vertex partition (also known as node separator), that is, one wants to remove a small number of vertices such that the graph splits into two components. Although there are methods of converting an edge separator into a vertex separator and vice versa (e.g., Ref. [22]), the quality of the solutions after the conversion can be very poor. Nevertheless, a good vertex separator might serve as a starting point of a search for a good edge separator.

An additional extension is the repartitioning problem. In this case, an unbalanced partition π_t of a graph is present and the task is to find a nearby remapping π_{t+1} of the vertices that restores the balance. This usually results in a multiobjective optimization problem since most applications not only request few cut edges but also a small number of migrating vertices $|\{v \in V : \pi_t(v) \neq \pi_{t+1}(v)\}|$. Some existing approaches addressing this problem can be found, for example, in Refs. [23,24].

60.1.2 Bounds on the Bisection Width

Some analytical results on the bisection width of graphs are known. Feige et al. [25] propose an algorithm which calculates a cut-size that differs from the bisection width by not more than a factor of $\mathcal{O}(\sqrt{|V|} \cdot \log(|V|))$. Although this factor is still very high, it is the first sublinear factor for this problem. Furthermore, an algorithm with a smaller factor of $\mathcal{O}(\log^2(|V|))$ has been proposed in Ref. [26]. These approximation factors are of high theoretical interest, but they are far from acceptable for real applications. Furthermore, the algorithms behind these approximation factors are very complicated and are not suitable to design fast and efficient graph partitioning algorithms.

The graph bisection problem is **NP**-complete even for regular degree graphs [27]. Analytical results on these graphs show that almost every large d-regular graph $G = (V, E)$ has a bisection width of at least $c_d \cdot |V|$ where $c_d \rightarrow \frac{d}{4}$ as $d \rightarrow \infty$ [28,29].

These bounds can be improved for small values of d. Almost every large 3-regular graph has a bisection width of at least $\frac{1}{9.9}|V| \approx 0.101|V|$ [30,31]. However, all sufficiently large 3-regular graphs have a bisection width of at most $\frac{1}{6}|V|$ [16]. Furthermore, it is shown that almost all large 4-regular graphs have a bisection width of at least $\frac{11}{50}|V| = 0.22|V|$ [29], while the bisection width of sufficiently large 4-regular graphs is at most $\frac{2}{5}|V|$.

Some approaches calculate lower bounds on the graph bisection width. These bounds can be used to evaluate the quality of the existing upper bounds as well as to speed up Branch & Bound strategies determining the bisection width of moderately sized graphs. Leighton [32] proposes a lower bound of the bisection width based on a routing scheme for all pairs of vertices. A small congestion of the routing scheme leads to a high lower bound. Lower bounds on the bisection width can also be derived from algebraic graph theory by relating the bisection problem to an eigenvalue problem. It is well known that the bisection width of a graph $G = (V, E)$ is at least $\frac{\lambda_2|V|}{4}$ with λ_2 being the second smallest eigenvalue of the Laplacian of G. This spectral bound is tight for some graphs [33].

Furthermore, the structure of an optimal bisection can be used to derive improved spectral lower bounds on certain graph classes [33]. For some classes of d-regular graphs one can prove an improved lower bound on the bisection width of roughly $\frac{d}{d-2}\frac{\lambda_2|V|}{4}$. Furthermore, one can prove a lower bound of $\frac{10+\lambda_2^2-7\lambda_2}{8+3\lambda_2^3-17\lambda_2^2+10\lambda_2} \cdot \frac{\lambda_2|V|}{2}$ for the bisection width of all sufficiently large 3-regular graphs and a lower bound of $\frac{5-\lambda_2}{7-(\lambda_2-1)^2} \cdot \frac{\lambda_2|V|}{2}$ for the bisection width of all sufficiently large 4-regular graphs. These lower bounds are higher than the classical bound of $\frac{\lambda_2|V|}{4}$ for sufficiently large graphs and are applicable to Ramanujan graphs [34–37]. Any sufficiently large 3-regular Ramanujan graph has a bisection width of at least $0.082|V|$, while sufficiently large 4-regular Ramanujan graphs have a bisection width of at least $0.176|V|$. These values are the best lower bounds for explicitly constructible 3- and 4-regular graphs [16].

60.1.3 Applications

The graph partitioning problem is encountered, for example, in Very-Large-Scale Integration (VLSI) layout of circuits. The wires of the circuit have to be distributed uniformly over a wafer and long wire lengths should be avoided. This problem can be solved by recursive partitioning of the graph which represents the circuit such that the costs between two partitions are minimized. In Ref. [38] it is reported that this method is the most cost-effective way to generate VLSI placements.

Another application consists in balancing the computational load in distributed (adaptive) finite element method (FEM) computations. These are used extensively by engineers to analyze a variety of physical processes which can be expressed via partial differential equations (PDE). The domain on which the PDEs have to be solved is discretized into a mesh, and the PDEs are transformed into a set of linear equations defined on the mesh's elements (see, e.g., Ref. [39]). These can then be solved by iterative methods such as the conjugate gradient (CG). Due to the very large amount of elements needed to obtain an accurate approximation of the original problem, this method has become a classical application for parallel computers. The parallelization of numerical simulation algorithms usually follows the single-program multiple-data (SPMD) paradigm: Each processor executes the same code on a different parts of the data. This means that the mesh has to be split into P subdomains and each subdomain is then assigned to one of the P processors. To minimize the overall computation time, all processors should roughly contain the same amount of elements. Since iterative solution algorithms perform mainly local operations, that is, data dependencies are defined by the mesh, the parallel algorithm mainly requires communication at the partition boundaries. Hence, these should be as small as possible. Since the communication pattern of FEM computations can be modeled by a graph where the vertices represent the data and the edges the dependencies, this task is equivalent to solving a graph partitioning problem.

60.2 Global Heuristics

Apart from simple greedy algorithms, there are only a few global heuristics that are still applied to partition a graph. Most of these heuristics are not competitive to the multilevel schemes described later on. Therefore, we only mention two popular approaches here.

The first is the spectral bisection. Note that, though we describe only spectral bisection, this heuristic has been also generalized to solve k-partitioning problems directly within a multilevel framework which is described later on. Spectral bisection does not work directly on a given graph, but on a mathematical representation. For a graph $G = (V, E)$ with vertex-edge incidence matrix $\mathbf{A}$, which contains in each column corresponding to edge $e = (u, v)$ the entries -1 and $+1$ in the rows u and v, and 0 elsewhere. The Laplacian matrix $\mathbf{L} \in \mathbb{Z}^{|V|\times|V|}$ of G is defined as $\mathbf{L} = \mathbf{A}\mathbf{A}^{\mathrm{T}}$. For the spectral bisection, the second-smallest eigenvalue λ_2 of $\mathbf{L}$ is computed. Then, the median m of all components of the corresponding eigenvector e is determined and the vertices of the graph are distributed as $V_1 = \{v \in V : e_v < m\}$ and $V_1 = \{v \in V : e_v > m\}$. More background information on this method can be, for example, found in Ref. [40]. Also, extensions to k-partitioning have been published [41].

In some applications vertices are provided with geometric data. Hence, this additional information can be used to partition the graph. In this field, partitions based on space-filling curves have become popular. Space-filling curves are geometric representations of bijective mappings $M : \{1, \dots, N^m\} \rightarrow \{1, \dots, N\}^m$. The curve M traverses all N^m cells in the m-dimensional grid of size N. They have been introduced by Peano and Hilbert in the late nineteenth century [42]. An (historic) overview on space-filling curves is given in Ref. [43]. In Ref. [44], it is shown that partitions based on connected space-filling curves are "quasi optimal" for regular grids and special types of adaptively refined grids. In these cases, the cut-size is bounded by $C(|V|/P)^{(d-1)/d}$, where $|V|$ denotes the number of vertices, P the number of partitions, and d the dimension of the graph. The constant C depends on the curve type. This approach is very fast, but it is also known that ignoring the adjacency information of the graph can result in a poor solution quality, especially in case of unstructured graphs.

60.3 The Multilevel Graph Partitioning Paradigm

The graph partitioning problem often only represents a subproblem. Therefore, it has to be solved fast and be space-efficient. Due to the size of the graphs, global approaches are either slow or result in rather bad solutions. This is overcome by the most important invention in this area, the multilevel scheme (see, e.g., Ref. [1]).

60.3.1 The Multilevel Scheme

The main idea of the multilevel scheme is to contract vertices of the graph and generate a new level consisting of a smaller graph with a similar structure. This is repeated, until in the lowest level only a small graph, sometimes with two vertices only, remains. The partitioning problem is then solved for this small graph and vertices in higher levels are assigned to partitions according to their representatives in lower levels, after a local refinement phase has been applied to further enhance the current solution. This process finally leads to a partition of the original graph. Hence, a multilevel algorithm consists of three important tasks: (i) a matching algorithm, deciding which vertices are combined in the next level, (ii) a global partitioning algorithm applied in the lowest level (which actually can be omitted if the number of vertices in the lowest level meets the number of desired partitions), and (iii) a local refinement algorithm improving the quality of a given partition.

Edges of high weights usually connect dense areas of the graph. To impede these edges from being cut in coarser graphs, the matching algorithm is supposed to calculate a solution with a high edge weight. To speed up the coarsening process, the whole graph is processed and the matchings on each level should have a high cardinality. The maximum reduction is achieved by splitting the number of vertices in halves on each level. This is only possible with a complete matching. Section 60.3.2 therefore presents a number of approximation algorithms for maximum-weighted matchings in general edge-weighted graphs.

Concerning the local improvement, most heuristics are based on the work of Kernighan and Lin [12]. Their strategy is to move single vertices between parts to decrease the size of the cut. The Helpful-Set heuristic [13,14] is based on a movement of larger sets of vertices and is derived from a technique used to prove an upper bound on the bisection width of 4-regular graphs [15]. Both will be presented in Section 60.3.3.

60.3.2 Graph Coarsening through Graph Matching

To create a smaller, similar graph for the next level of the multilevel scheme, a matching algorithm is applied and matched vertices are then combined to generate the next level.

60.3.2.1 Graph Matching Algorithms

Graph matching is a fundamental topic in graph theory. Let $G = (V, E)$ be a graph consisting of the vertex set V and a set of undirected edges E without multiedges or self-loops. A matching of G is a subset $M \subset E$, so that no two edges of M are adjacent. A vertex incident to an edge of M is called *matched* and a vertex not incident to an edge of M is called *free.*

In the past, an enormous amount of work has been spent on matching theory. Different types of matchings have been discussed, their existence and properties have been analyzed, and efficient algorithms to calculate specific matchings have been developed. Many results have been achieved for specific classes such as bipartite or planar graphs. A central aspect are matchings with high cardinality. A *Maximal Matching* M_{MAX} is a matching which cannot be enlarged by an additional edge without violating the matching property. A graph may have several different maximal matchings and, especially, maximal matchings of different cardinality. A *Maximum Cardinality Matching* M_{MCM} is a matching of maximum size, that is, for all matchings $\bar{M}$ of G it holds $|M_{\mathsf{MCM}}| \geq |\bar{M}|$.

Matchings are also discussed for graphs with edge weights $w : E \to I\!R$. For a set $F \subset E$ let $W(F) := \sum_{\{a,b\}\in F} w(\{a, b\})$ be the weight of F. A *maximum-weighted Matching* M_{MWM} is a matching of highest weight, that is, for all matchings $\bar{M}$ of G it holds $W(M_{\mathsf{MWM}}) \geq W(\bar{M})$.

Many algorithms calculating matchings have been developed in the past. Consult, for example, Refs. [45, 46] for the history of matching algorithms. In the following, only the currently fastest algorithms are stated.

The fastest algorithm for maximum cardinality matching up to date is presented in Ref. [47] and has a time complexity of $\mathcal{O}(|E|\sqrt{|V|})$. In the edge-weighted case, the algorithm from Ref. [48] calculates a maximum-weighted matching in time $\mathcal{O}(|V| \cdot |E| + |V|^2 \log(|V|))$. This time complexity has been improved by Gabow and Tarjan [49] considering the assumption of integral costs which are not particularly high: if the weight function $w : E \to [-N, ..., N]$ maps to integers between $-N$ and N only, their algorithm will run in $\mathcal{O}(\sqrt{|V| \cdot \alpha(|E|, |V|) \cdot \log(|V|)} \cdot |E| \cdot \log(|V| \cdot N))$ time, with α being the inverse of Ackermann's function.

60.3.2.2 Approximation Algorithms for Cardinality Matching

The algorithms discussed so far have super-linear time complexity. Recently, approximation algorithms for matching problems have attracted more and more attention. They not only possess a smaller time complexity than optimal algorithms but also calculate suboptimal solutions. The guaranteed quality is described by an approximation factor which states the worst-case loss to an optimal solution, for example, a factor of $\frac{1}{2}$ guarantees that the solution quality is at least half as good as the optimum. It is a simple exercise to prove that any maximal matching M_{MAX} has a cardinality of at least $\frac{1}{2}$ the cardinality of a maximum cardinality matching, that is, $|M_{\mathsf{MAX}}| \geq \frac{1}{2}|M_{\mathsf{MCM}}|$. Therefore, any maximal matching algorithm is an $\frac{1}{2}$-approximation algorithm for a maximum cardinality matching. Simple methods with time complexity $\mathcal{O}(|E|)$ can be used to calculate maximal matchings.

Augmenting paths are often considered for graph matching, especially for approximating maximum cardinality matching. An augmenting path has an odd number of edges with alternating edges of M and of $E\backslash M$ and two free vertices as endpoints, that is, an augmenting path of length l consists of $\frac{l-1}{2}$ edges of M and $\frac{l+1}{2}$ edges of $E\backslash M$. If such a path exists, the cardinality of M can be increased by one by exchanging the matched and unmatched edges of the path.

Based on Ref. [50], it can be shown that if the shortest augmenting path with respect to a matching M_l has length l, then $|M_l| \geq \frac{l-1}{l+1}|M_{\mathsf{MCM}}|$ (see, e.g., Ref. [51, p. 156]). Matchings without short augmenting paths can be calculated very fast, for example, a matching with a shortest path of length $l \geq 5$ can be computed in time $\mathcal{O}(|E|)$, resulting in $|M_5| \geq \frac{2}{3}|M_{\mathsf{MCM}}|$. If the minimum degree min and maximum degree max of all vertices are considered, it can easily be proven that if the shortest augmenting path has a length of $l \geq 5$, then $|M_5| \geq \frac{min}{max+2 \cdot min}|V|$, which implies $|M_5| \geq \frac{1}{3}|V|$ for graphs with a regular degree, that is, $\frac{2}{3}$ of the vertices are matched [17].

60.3.2.3 Common Graph Matching Heuristics

A matching algorithm for multilevel partitioning is supposed to quickly deliver a high matching cardinality and matching weight. Because of the time constraints, the calculation of a M_{MCM} or even a M_{MWM} would be too time consuming. Therefore, fast algorithms calculating maximal matchings are applied. All of them follow the same strategy: Starting with an initial empty matching, the vertices of the graph are visited in a specific order. For each visited vertex v, it is checked if v is free and if v is adjacent to at least one free vertex. If v is free and all neighbors are already matched, v remains free. Otherwise, if v is free and at least one free neighbor exists, the edges to free neighbors are rated and an edge with highest rating is added to the matching. The methods differ in the order in which the vertices are visited and the rating of the incident edges. In addition, they may also differ in the way in which possible ties in the ordering and rating are broken.

As first example, the *random-edge* matching (REM) is used in, for example, Ref. [1]. The vertices are visited in random order, a random free neighbor is chosen and the connecting edge is added to the matching. In Refs. [2,52], the random matching as well as the *heavy-edge* matching (HEM), the *modified heavy-edge* matching (HEM*), the *light-edge* matching (LEM), and the *heavy-clique* matching (HCM) are discussed. All of them visit the vertices in random order and either take any incident edge with highest weight (HEM), perform further tie-breaking mechanisms if more than one edge has the highest weight (HEM*), take the edge with lightest weight (LEM), or take the edge with the highest *edge-density*. The first attempt to analyze the matching strategies REM and HEM under several assumptions has been published in Ref. [53]. The work in Ref. [3] proposes a *balanced-edge* matching to result in a coarse graph with more uniform vertex weights.

These strategies are fast and try to calculate a matching with high cardinality and weight but only guarantee a maximal matching. This, as discussed above, guarantees a cardinality of the matching with at least $\frac{1}{2}|M_{\mathsf{MCM}}|$. However, concerning the matching weight, one can construct examples for which these methods calculate matchings with a weight much lower than that of a M_{MWM}.

An experimental study of several matching algorithms is given in Ref. [54]. An overview of parallel algorithms for graph matching problems is presented in Ref. [51] and new parallel approximation algorithms are presented in Ref. [55]. In the following we present a number of fast sequential approximation algorithms for graph matching.

60.3.2.4 $\frac{1}{2}$-Approximation Algorithms for Maximum-Weighted Matching

There is a series of different $\frac{1}{2}$-approximation algorithms, starting with a very simple algorithm which is based on a greedy strategy. It always adds the heaviest free edge to the matching [56].

Greedy algorithm

```
$M_{Greedy} := \emptyset$;
Sort the edges $E$ according to their weight;
WHILE ($E \neq \emptyset$)
    take an edge $\{a, b\} \in E$ with highest weight;
    add $\{a, b\}$ to $M_{Greedy}$;
    remove all edges incident to $a$ or $b$ from $E$;
```

The Greedy algorithm, which is, for example, applied in Ref. [4] for calculating matchings after the first few coarsening steps, requires a time of $\mathcal{O}(|E| \cdot \log(|V|))$ because of the initial sorting of the edges by their weight. Clearly, if the Greedy algorithm chooses an edge which is not part of an (unknown) maximum-weighted matching, the chosen edge blocks at most two edges of the optimal matching. However, the edge weight of each of the two blocked edges are at most the weight of the chosen edge.

Theorem 60.1 (**due to Avis** [56])

Let $G = (V, E)$ be a graph with vertices V and weighted undirected edges E. The Greedy algorithm computes a matching of G with an edge weight of at least $\frac{1}{2}$ of the edge weight of a maximum-weighted matching in time $\mathcal{O}(|E| \cdot \log(|V|))$.

The work in Refs. [7,8] started a sequence of new $\frac{1}{2}$-approximation algorithms for maximum-weighted matching by different authors. They all require linear time.

Locally Heaviest Algorithm

```
  $M_{Locally-Heaviest} := \emptyset$;
  WHILE ($E \neq \emptyset$)
      try match ($\{a, b\}$) with an arbitrary edge $\{a, b\} \in E$;
PROCEDURE try match ($\{a, b\}$)
  WHILE ($a$ is free AND $b$ is free AND there is another edge $\{a, c\} \in E$ or $\{b, d\} \in E$ with a higher
      weight)
      IF ($a$ is free AND there is another edge $\{a, c\} \in E$ with a higher weight)
          try match ($\{a, c\}$);
      IF ($b$ is free AND there is another edge $\{b, d\} \in E$ with a higher weight)
          try match ($\{b, d\}$);
  IF ($a$ and $b$ are free)
      add $\{a, b\}$ to $M_{Locally-Heaviest}$;
      remove all edges incident to $a$ or $b$ from $E$;
```

This algorithm starts with an empty matching $M_{Locally-Heaviest}$ and repeatedly chooses an arbitrary edge which it tries to match. The main idea is to look for the so-called *locally heaviest* edges, that is, edges which are at least as heavy as all of their remaining adjacent edges. After such an edge has been matched and removed from E, further remaining edges may become locally heaviest. Clearly, if one repeatedly matches locally heaviest edges, the resulting matching will be a $\frac{1}{2}$ approximation of the maximum-weighted matching because of the same fact as with the Greedy algorithm. The procedure try match proceeds in a backtracking manner. As long as there exists an incident edge to one of the vertices a or b with a higher weight than $\{a, b\}$, it calls itself recursively with this new edge. This is repeated until all adjacent edges have a lower or equal weight. In this case, the current edge is a locally heaviest edge and is added to the matching while all adjacent edges are removed.

Theorem 60.2 (**due to Monien et al.** [7])

Let $G = (V, E)$ be a graph with vertices V and weighted undirected edges E. The Locally Heaviest algorithm computes a matching of G with an edge weight of at least $\frac{1}{2}$ of the edge weight of a maximum-weighted matching in linear time $\mathcal{O}(|E|)$.

The path growing algorithm from Ref. [9] possesses the same approximation quality and run-time as the Locally Heaviest algorithm presented before but its analysis is much simpler.

Path-Growing Algorithm

```
start with an empty set of paths $P$;
WHILE (there is a vertex $v \in V$)
    WHILE ($v$ has a neighbor)
        let $\{v, w\}$ be a heaviest edge incident to $v$;
        add $\{v, w\}$ to the set of paths $P$;
        remove $v$ from $V$;
        $v := w$;
divide $P$ into two matchings $M_1$ and $M_2$ and return the heavier one;
```

Its main idea is to grow a set of vertex disjoint paths. Starting at a single vertex, it tries to extend the current path in one direction always along the heaviest edge available. Once the heaviest edge has been chosen, all other edges incident to the current vertex are removed. If no more edges can be found to extend the current path, a new path is started at a pathless vertex. This process is repeated until no more pathless vertices remain. The edges of all paths are separated alternately in two sets M_1 and M_2. Therefore, both sets do not contain any incident edges and are both matchings. Finally, the heavier set (matching) is returned.

It is easy to see that the algorithm runs in linear time, since each vertex is processed only once and finding all heaviest incident edge can be executed in $\mathcal{O}(|E|)$. Since the algorithm always chooses the heaviest incident edge, the weight of $M_1 \cup M_2$ is at least as large as the weight of MWM and hence the heavier set has at least half the weight. This results in the following theorem.

Theorem 60.3 (due to Drake and Hougardy [9])

Let $G = (V, E)$ be a graph with vertices V and weighted undirected edges E. The Path-Growing algorithm computes a matching of G with an edge weight of at least $\frac{1}{2}$ of the edge weight of a maximum-weighted matching in linear time $\mathcal{O}(|E|)$.

60.3.2.5 $\frac{2}{3} - \epsilon$ Approximation Algorithms for Maximum-Weighted Matching

It is possible to design an approximation algorithm with an approximation factor arbitrarily close to $\frac{2}{3}$ [10]. The algorithm iteratively improves a matching via short augmentations to achieve an approximation factor of $\frac{2}{3} - \epsilon$.

Iterated-Augmentation Algorithm

```
compute a $\frac{1}{2}$-approximation matching $M$ in linear time (previous section);
DO at most $\frac{16}{3\epsilon}$ iterations of
    $M' := M$;
    FOR ($e \in M$) DO
        IF (there exists a $\beta$-augmentation in $M'$ with center $e$)
            augment $M'$ by a good $\beta$-augmentation with center $e$;
    $M := M'$;
```

This algorithm starts with a matching computed by a linear time $\frac{1}{2}$ matching algorithm (Locally Heaviest or Path-Growing). The key idea is to apply short augmentations, that is, augmentations around a center edge e involving only edges adjacent to the center edge. Furthermore, no arbitrary short augmentations are performed, but only *good* augmentations that yield a certain ratio. The parameter β controls this selection and depends only on ϵ. It can be proven that a constant number of improvement iterations (depending on ϵ) is sufficient to achieve the desired approximation. Overall, the following theorem can be stated.

Theorem 60.4 (due to Drake and Hougardy [10])

Let $G = (V, E)$ be a graph with vertices V and weighted undirected edges E. For any constant $\epsilon > 0$ the Iterated-Augmentation algorithm computes a matching of G with an edge weight of at least $\frac{2}{3} - \epsilon$ of the edge weight of a maximum-weighted matching in linear time $\mathcal{O}(|E|)$.

Another $\frac{2}{3} - \epsilon$ approximation algorithm is described in Ref. [11]. It converges faster toward the desired approximation than the Iterated-Augmentation algorithm.

Deterministic-Match Algorithm

```
$M := \emptyset$;
REPEAT $k$ times
    $S := \emptyset$;
    FOREACH atom $e$ (matched edge or unmatched vertex)
        find an eligible augmentation $a$ centered at $e$ maximizing the gain;
        $S := S \cup \{a\}$;
    calculate a vertex disjoint subset of augmentations from $S$ and apply them to $M$;
```

Like the Iterated-Augmentation algorithm, this approach also requires a constant number of iterations. The number of iterations k only depends on the desired gap of the approximation factor to $\frac{2}{3}$. In each iteration, every atom (either a matched edge or an unmatched vertex) is choosing a small augmentation.

This results in a set S of augmentations, which are not necessarily independent. Then, by repeatedly selecting edges with the highest gain in a greedy fashion, a maximum vertex-disjoint augmentation in S is constructed. This is then realized on M. The properties of the algorithm are stated in the following theorem.

Theorem 60.5 (due to Pettie and Sanders [11])

Let $G = (V, E)$ be a graph with vertices V and weighted undirected edges E. For any constant integer k the Deterministic-Match algorithm computes a matching of G with an edge weight of at least $\frac{2}{3}(1 - (\frac{19}{20})^k)$ of the edge weight of a maximum-weighted matching in time $\mathcal{O}(k|E|)$.

60.3.2.6 Heuristic Improvements

Without modifications, the presented matching algorithms will often create star-like graphs, meaning that some very heavy vertices of very high degree are generated (see, for example, Ref. [17]). This makes the result unusable because neither is the graph similar to the original one anymore nor can the number of vertices of a star-like graph be reduced appropriately to create a smaller graph for the next level of about half the size. To prevent this, the edge-weights can be modified according to the weight of their incident vertices. The new weight w' of an edge $e = \{u, v\}$ becomes $w'(e) = w(e)/(w(u) \cdot w(v))$. Another possibility is to only consider vertices as matching pairs if their combined weight does not exceed twice the weight of the lightest plus the weight of the heaviest vertex that occurs in the entire graph. This ensures that heavy vertices can only be matched with light ones which leads to more balanced weights. Further discussion can be found, for example, in Ref. [17].

60.3.3 Local Refinement Algorithms

The second important phase within the multilevel paradigm is the improvement or local refinement. After the vertices are assigned according to their representatives in the smaller graph, this phase tries to improve the current partitioning.

60.3.3.1 The Kernighan–Lin Refinement Heuristic

The KL heuristic [12] with the modifications by Fiduccia and Mattheyses [57] is one of the earliest graph partitioning heuristics and has been developed to optimize placements of electronic circuits. It does not create partitions but rather improves existing ones by exchanging vertices.

Each vertex v of the graph is assigned a value diff(v) which represents the reduction in the chosen metric that would occur when moving v to the other partition. If addressing the cut-size, this value depends only on the edges incident to v and is defined as

$$\text{diff}(v) = \sum_{\{v,u\} \in E; \pi(v) \neq \pi(u)} w_E(\{v, u\}) - \sum_{\{v,u\} \in E; \pi(v) = \pi(u)} w_E(\{v, u\})$$

Also, if a vertex is moved, only the values of itself and its neighbors are affected.

The KL algorithm performs several passes. In each pass, it repeatedly chooses an unlocked vertex with a high value, moves it logically to the other part and locks it. If the resulting bisection after a move is balanced and the cut-size is lower than the previously found best cut-size, it marks this state. When all vertices are locked and if a new best cut-size has been marked, all vertex moves up to the marked state are performed physically. The algorithm continues with the next pass until the cut-size cannot be reduced further.

Kernighan–Lin Algorithm

```
REPEAT
    compute the diff-values of all vertices
    WHILE there are unlocked vertices
        IF the partition is not balanced
            choose an unlocked vertex $v$ with maximal diff-value from the overloaded part
        ELSE
```

```
            choose an unlocked vertex $v$ with maximal diff-value from any part
        logically move v to the other part and lock it
        update the diff-values of $v$'s neighbors
        IF the bisection is balanced and the cut-size is lower than the best cut-size seen so far
            mark the current state
    ENDWHILE
    IF marked cut-size is smaller than previously known cut-size
        move all vertices up to the marked state physically to the other part
UNTIL cut-size is not reduced
```

The original algorithm by Kernighan and Lin is based on the exchange of vertex pairs. In a simple implementation, a pass has run-time $\mathcal{O}|V|^3$ with the calculation of the vertices with the highest gain being the most expensive part. By sorting the vertices according to their gain, this can be reduced to $\mathcal{O}(|V|^2 \log|V|)$. The method has been modified in Ref. [57], such that only single vertices are moved. Furthermore, efficient data structures called *buckets* reduce the run-time per pass to $\mathcal{O}(|V| + |E|)$.

In general, KL is robust and reliable. Although no theoretical bounds on the partition quality is known yet, the results are convincing, provided the initial partitioning is fairly satisfactory.

60.3.3.2 The Helpful-Set Refinement Heuristic

Just as KL, the Helpful-Set bisection heuristic [13,14] is based on local search. Starting with an arbitrary initial bisection π, it tries to reduce the cut-size with the help of local rearrangements. However, their choice is the main difference to KL since it does not only migrate single vertices but also sets off vertices.

The Helpful-Set concept is based on the following definition which states the reduction of the cut-size if a set of vertices is moved from one part to another.

Definition 60.1 (Helpful-Sets)

Let π be a bisection of the vertices V of a graph $G = (V, E)$ into V_1 and V_2. For a set $S \subset V_i$, $i \in \{1, 2\}$, let

$$H(S) = |\{\{v, w\} \in E; v \in S; w \in V \backslash V_i\}| - |\{\{v, w\} \in E; v \in S; w \in V_i \backslash S\}|$$

be the helpfulness *of S. S is called $H(S)$*-helpful.

Helpful-Set Algorithm

```
REPEAT
    search for a $k$-helpful set $S \subset V_1$ with $k > 0$ and move it to $V_2$;
    search for a $\bar{k}$-helpful balancing set $\bar{S} \subset V_2$ with $|S| = |\bar{S}|$ and $k + \bar{k} > 0$ and move it to $V_1$;
UNTIL a desired cut-size is reached
```

The algorithm proceeds within several iterations. Each iteration starts to search for a k-helpful set $S \subset V_1$ with a positive k. Then, an equally sized balancing set $\bar{S}$ will be found on the enlarged part and moved back to the first part. The helpfulness of the balancing set can be negative, but the overall gain will be positive. This ensures that the cut-size decreases in each iteration.

The idea behind this process is that as long as the cut-size is above a certain value, one can guarantee that a small k-helpful set can be found on one side and a balancing set on the other side. The first theorem based on this observation is the following.

Theorem 60.6 (Due to Hromkovič and Monien [15])

Let $G = (V, E)$ be a 4-regular graph with an even number of at least 350 vertices. Then, the bisection width $\mathrm{bw}(G)$ *of G is bounded by*

$$\mathrm{bw}(G) \leq \frac{1}{2}|V| + 1$$

The proof of this theorem is based on two lemmas, one for finding a helpful set in one of the parts and one for balancing an almost balanced partition without increasing the cut-size too much.

Lemma 60.1 **(Helpful Set; due to Hromkovič and Monien [15])**

Let π be a balanced partition with $cut(\pi) > \frac{|V|}{2} + 1$. Then we can find a balanced partition $\hat{\pi}$ with $cut(\hat{\pi}) < cut(\pi)$ by exchanging only two nodes or there exists a 4-helpful graph of cardinality at most $4\lceil \log_2 |V| \rceil + 3$.

Clearly, this lemma can be applied if the current cut size is still too high. Then, either the cut size is reduced by a simple exchange of two vertices, or it is decreased by at least four if the small 4-helpful set is moved to the other part.

We will call a partition π *almost balanced* if there exists some number z such that (i) $cut(\pi) \geq z|\frac{|V|}{2} - |V_1||$ and (ii) $4\max\{|V_1|, |V_2|\} \leq 5cut(\pi) + 2z$.

Lemma 60.2 **(Balancing Lemma, due to Hromkovič and Monien [15])**

Let π be an almost balanced partition with $cut(\pi) \geq 4$. Then we can construct a balanced partition $\hat{\pi}$ with $cut(\hat{\pi}) \leq cut(\pi) + 2$.

Thus, the first lemma ensures a reduction of the cut-size of at least 4 and the second lemma ensures an increase of at most 2, leading to an overall reduction of 2 for every iteration of these two lemmas. This results in the already stated theorem.

These results are extended to general regular graphs with an even degree.

Theorem 60.7 **(Due to Monien and Diekmann [14])**

Let $G = (V, E)$ be a d-regular graph with d even, $d \geq 4$ and $|V| \geq n_0(d)$. Then, the bisection width $\mathrm{bw}(G)$ *of G is bounded by*

$$\mathrm{bw}(G) \leq \frac{d-2}{4}|V| + 1$$

Furthermore, improved results are known for 3- and 4-regular graphs.

Theorem 60.8 **(Due to Monien and Preis [16])**

For any $\epsilon > 0$ there exists a value $n(\epsilon)$ such that the bisection width of any 3-regular graph $G = (V, E)$ with $|V| > n(\epsilon)$ is at most $(\frac{1}{6} + \epsilon)|V|$.

Theorem 60.9 **(Due to Monien and Preis [16])**

For any $\epsilon > 0$ there exists a value $n(\epsilon)$ such that the bisection width of any 4-regular graph $G = (V, E)$ with $|V| > n(\epsilon)$ is at most $(\frac{2}{5} + \epsilon)|V|$.

All proofs of these theorems are based on the Helpful-Set concept. Although these results are stated for regular graphs and only for certain values of regularity, they can be generalized in principal to a wide range of vertex degrees. However, these proofs would be very technical.

The Helpful-Set heuristic [7,13] is the algorithmic result of the theorems above and includes several heuristic generalization such as the handling of graphs with arbitrary vertex degrees and graphs with vertex and edge weights.

60.3.4 Graph Partitioning Tools

Efficiency and generalizations of graph partitioning methods strongly depend on specific implementations. There exist several software libraries which provide a large range of different methods. Examples are CHACO [58], JOSTLE [59], METIS [60], PARTY [7,21], and SCOTCH [61]. The goal of the libraries is both to provide efficient implementations and to offer a flexible and universal graph partitioning interface to applications.

Although there exists a large number of sequential libraries, only a few parallel implementations are available. This is because of the complexity involved in parallel programming. Furthermore, the applied heuristics like KL are basically of sequential nature, hence modifications are required, which sometimes introduce new limitations. The most popular distributed libraries are the parallel versions of METIS [60,62] and JOSTLE [63,64]. These tools essentially perform the same techniques as their sequential counterparts, are quite fast and deliver solutions that are acceptable for most applications.

60.3.5 Effects of Nondeterminisms

Each of the libraries implementing the algorithms presented above involves some kind of heuristic. This is unavoidable due to the complexity of the problem, the large problem sizes, and the given time constraints. Even if approximation algorithms are applied in some stages, the overall computation is still a heuristic, because the influence between the different components has not been theoretically investigated yet. However, this also means that empirical tests are necessary and are currently the only way to compare the different approaches and implementations.

A number of example graphs from different sources have been published. Library authors usually refer to a subset of these graphs to evaluate their algorithms and measure the improvements. However, observations in Ref. [65] reveal that different initializations of the random number generator, which is applied, for example, in the matching algorithm of some libraries, lead to a large variety in the quality of the results.

Schamberger [66] demonstrates that results calculated by partitioning libraries are not totally predictable, even if the applied algorithms themselves do not include any nondeterminism. By generating graphs of identical structure but with a permuted vertex ordering, one can observe that the obtained partitions vary significantly in their quality. The variation is sometimes even larger than the gap between different implementations.

This is important since experiments and comparisons are usually thought as a kind of prediction of how good a heuristic performs on certain types of graphs. Randomization also helps during the development and testing of new features. Additionally, it provides information about how reliable a graph partitioning heuristic is and therefore indicates in what quality range solutions can be expected.

60.4 Discussion

The existing graph partitioning heuristics provide good solutions and are very fast. However, although great progress has been made in this area, many questions and challenges remain as, for example, explained in Ref. [67]. While the global cut-size is the classical metric that most graph partitioners optimize, it is not necessarily the metric that models the real costs of the application. In FEM computations for example, the true communication volume can significantly differ from the number of cut-edges. Furthermore, aspects like latency are ignored. Another questionable point is the applied norm. In synchronized computations, the slowest processor specifies the overall speed, hence the maximum norm would be appropriate, while the usually applied cut-size is a summation norm.

Furthermore, the existing libraries cannot obey some constraints that applications might have. For example, the multilevel algorithms cannot guarantee connected partitions. Swapping a single vertex in a lower level can easily disconnect a partition. Though it is possible to check the result for connectivity, it would be quite an expensive operation and therefore is usually not implemented. Another point especially when solving the repartitioning problem is that partitions often become long and thin. This contradicts the desired small boundaries and, for example, in FEM computations, can even lead to instabilities in the parallel mathematical solvers. However, there are first approaches that consider these additional constraints.

Another concern is the parallelization. All state-of-the-art implementations are based on vertex exchange heuristics like KL. This procedure, however, is a sequential process. Although it is possible to parallelize it by adding some restrictions, the question remains if there exists a method that is of parallel nature. With growing graph sizes, large clusters and with the upcoming multicore processors, parallelization becomes more and more important.

References

[1] Hendrickson, B. and Leland, R., A multilevel algorithm for partitioning graphs, *Supercomputing '95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, ACM Press, New York, NY, 1995, p. 28.

[2] Karypis, G. and Kumar, V., A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM J. Scientific Comp.*, 20(1), 359–392, 1998.

[3] Karypis, G. and Kumar, V., Multilevel k-way partitioning scheme for irregular graphs, *J. Parallel and Dist. Comp.*, 48, 96, 1998.

[4] Gupta, A., Fast and effective algorithms for graph partitioning and sparse matrix reordering, *IBM J. Res. Dev.*, 41, 171, 1997.

[5] Ponnusamy, R., Mansour, N., Choudhary, A., and Fox, G. C., Graph contraction for mapping data on parallel computers: a quality-cost tradeoff, *Sci. Prog.*, 3, 73, 1994.

[6] Bouhmala, N., Impact of Different Graph Coarsening Schemes on the Quality of the Partitions, TR RT98/05-01, University of Neuchatel, Department of Computer Science, 1998.

[7] Monien, B., Preis, R., and Diekmann, R., Quality matching and local improvement for multilevel graph-partitioning, *Parallel Comp.*, 26(12), 1609, 2000.

[8] Preis, R., Linear time $\frac{1}{2}$-approximation algorithm for maximum weighted matching in general graphs, *Proc. STACS*, Lecture Notes in Computer Science, Vol. 1563, Springer, Berlin, 1999, p. 259.

[9] Drake, D. E. and Hougardy, S., Improved linear time approximation algorithms for weighted matchings, *Proc. APPROX/RANDOM*, Lecture Notes in Computer Science, Vol. 2764, Springer, Berlin, 2003, p. 14.

[10] Drake, D. E. and Hougardy, S., A simple approximation algorithm for the weighted matching problem, *Inf. Proc. Lett.*, 85(4), 211, 2003.

[11] Pettie, S. and Sanders, P., A simpler linear time $2/3-\epsilon$ approximation for maximum weight matching, *Inf. Proc. Lett.*, 91, 271, 2004.

[12] Kernighan, B. W. and Lin, S., An effective heuristic procedure for partitioning graphs, *Bell Syst. Tech. J.*, 291, 1970.

[13] Diekmann, R., Monien, B., and Preis, R., Using helpful sets to improve graph bisections, in *Interconnection Networks and Mapping and Scheduling Parallel Computations*, Vol. 21, Hsu, D. F., Rosenberg, A. L., and Sotteau, D., Eds., DIMACS Series in Disc. Math. and Theor. Comp. Sci., 1995, p. 57.

[14] Monien, B. and Diekmann, R., A local graph partitioning heuristic meeting bisection bounds, *SIAM Conf. on Parallel Proc. for Scientific Comp.*, Minneapolis, Minnesota, USA, 1997.

[15] Hromkovič, J. and Monien, B., The bisection problem for graphs of degree 4 (configuring transputer systems), in *Festschrift zum 60, Geburtstag von Günter Hotz*, Buchmann, J., Ganzinger, H., and Paul, W. J., Eds., 1992, p. 215.

[16] Monien, B. and Preis, R., Upper bounds on the bisection width of 3- and 4-regular graphs, *Proc. Int. Symp. on Mathematical Foundations of Comp. Sci.*, Lecture Notes in Computer Science, Vol. 2136, Springer, Berlin, 2001, p. 524. *J. Disc. Algorithms* (to appear).

[17] Preis, R., Analyses and Design of Efficient Graph Partitioning Methods, Dissertation, Universität Paderborn, Germany, 2000.

[18] Garey, M. R., Johnson, D. S., and Stockmeyer, L., Some simplified NP-complete graph problems, *Theor. Comp. Science*, 1, 237, 1976.

[19] Garey, M. R. and Johnson, D. S., *Computers and Intractability—a Guide to the Theory of NP-Completeness*, W. H. Freemann, San Francisco, CA, 1979.

[20] Simon, H. and Teng, S., How good is recursive bisection, *SIAM J. Sci. Comp.*, 18(5), 1436, 1997.

[21] Monien, B. and Schamberger, S., Graph partitioning with the party library: helpful-sets in practice, *Proc. Symp. on Comp. Arch. and High Perf. Comp.*, 2004, p. 198.

[22] Liu, J. W.-H., A graph partitioning algorithm by node separators, *ACM Trans. Math. Software*, 15(5), 198, 1989.

[23] Biswas, R. and Oliker, L., PLUM: parallel load balancing for adaptive unstructured meshes, *Parallel Dist. Comp.*, 51(2), 150, 1998.

[24] Biswas, R., Das, S. K., and Harvey, D. J., MinEX: a latency-tolerant dynamic partitioner for grid computing applications, *Future Generation Comp. Syst.*, 18(4), 477, 2002.

[25] Feige, U., Krauthgamer, R., and Nissim, K., Approximating the minimum bisection size, *Proc. STOC*, 2000.

[26] Feige, U. and Krauthgamer, R., A polylogarithmic approximation of the minimum bisection, *SIAM J. Comput.*, 31(4), 1090, 2002.

[27] Bui, T. N., Chaudhuri, S., Leighton, F. T., and Sisper, M., Graph bisection algorithms with good average case behaviour, *Combinatorica*, 7(2), 171, 1987.

[28] Clark, L. H. and Entringer, R. C., The bisection width of cubic graphs, *Bull. Australian Math. Soc.*, 39, 389, 1988.

[29] Bollobas, B., The isoperimetric number of random regular graphs, *Eur. J. Comb.*, 9, 241, 1988.

[30] Kostochka A. V. and Melnikov, L. S., On bounds of the bisection width of cubic graphs, *Proc. Czechoslovakian Symp. on Comb., Graphs and Complexity*, 1992, p. 151.

[31] Kostochka, A. V. and Melnikov, L. S., On a lower bound for the isoperimetric number of cubic graphs, *Proc. Intl. Conf. Probabilistic Methods in Disc. Math., Progress in Pure and Applied Disc. Math.*, Vol. 1, 1993, p. 251.

[32] Leighton, F. T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, San Francisco, CA, 1992.

[33] Bezrukov, S., Elsässer, R., Monien, B., Preis, R., and Tillich J.-P., New spectral lower bounds on the bisection width of graphs, *Theor. Comp. Sci.*, 320, 155, 2004.

[34] Chiu, P., Cubic Ramanujan graphs, *Combinatorica*, 12(3), 275, 1992.

[35] Lubotzky, A., Phillips, R., and Sarnak, P., Ramanujan graphs, *Combinatorica*, 8(3), 261, 1988.

[36] Margulis, G. A., Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators, *Probl. Inf. Transm.*, 24(1), 39, 1988.

[37] Morgenstern, M., Existence and explicit constructions of $q + 1$ regular Ramanujan graphs for every prime power q, *J. Comb. Theor., Ser. B*, 62(1), 44, 1994.

[38] Dutt, S., New faster Kernighan-Lin-type graph-partitioning algorithms, *Proc. Intl. Conf. on CAD*, 1993, p. 370.

[39] Fox, G., Williams, R., and Messina, P., *Parallel Computing Works!* Morgan Kaufmann, San Francisco, CA, 1994.

[40] Pothen, A., Simon, H., and Liou, K., Partitioning sparse matrices with eigenvectors of graphs, *J. Matrix Anal.*, 11, 430, 1990.

[41] Hendrickson, B. and Leland, R., An improved spectral graph partitioning algorithm for mapping parallel computations, *SIAM J. Sci. Comp.*, 16(2), 452, 1995.

[42] Hilbert, D., Über die stetige Abbildung einer Linie auf ein Flächenstück, *Mathematische Annalen*, 38, 459, 1891.

[43] Sagan, H., *Space Filling Curves*, Springer, Berlin, 1994.

[44] Zumbusch, G., *Parallel Multilevel Methods: Adaptive Mesh Refinement and Load Balancing*, Teubner, Wiesbaden, Germany, 2003.

[45] Lawler, E. L., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[46] Lovász, L. and Plummer, M. D., *Matching Theory*, Annals of Disc. Math., Vol. 29, North-Holland Mathematics Studies, Amsterdam, 1986.

[47] Micali, S. and Vazirani, V. V., An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs, *Proc. FOCS*, 1980, p. 17.

[48] Gabow, H. N., Data structures for weighted matching and nearest common ancestors with linking, *Proc. SODA*, 1990, p. 434.

[49] Gabow, H. N. and Tarjan, R. E., Faster scaling algorithms for general graph-matching problems, *JACM*, 38(4), 815, 1991.

[50] Hopcroft, J. and Karp, R. M., An $O(n^{5/2})$ algorithm for maximum matching in bipartite graphs, *SIAM J. Comp.*, 2, 225, 1973.

[51] Karpinski, M. and Rytter, W., *Fast Parallel Algorithms for Graph Matching Problems*, Oxford Lecture Series in Mathematics and its Applications, Vol. 9, Oxford University Press, Oxford, 1998.

[52] Karypis, G. and Kumar, V., Multilevel algorithms for multi-constraint graph partitioning, *Proc. Supercomputing Conf.*, IEEE Computer Society, Washington, DC, USA, 1998 pp. 1-13.

[53] Karypis, G. and Kumar, V., Analysis of multilevel graph partitioning, *Proc. Supercomputing Conf.*, ACM Press, New York, NY, USA, 1995, p. 29.

[54] Magun, J., Greedy matching algorithms, an experimental study, *J. Exp. Algorithmics*, 3, 6, 1998.

[55] Uehara, R. and Chen, Z.-Z., Parallel approximation algorithms for maximum weighted matching in general graphs, *Inf. Proc. Lett.*, 76, 13, 2000.

[56] Avis, D., A survey of heuristics for the weighted matching problem, *Networks*, 13, 475, 1983.

[57] Fiduccia, C. M. and Mattheyses, R. M., A linear-time heuristic for improving network partitions, *Proc. DAC*, 1982, p. 175.

[58] Hendrickson, B. and Leland, R., The Chaco User's Guide: Version 2.0, TR SAND94-2692, Sandia National Laboratories, Albuquerque, NM, 1994.

[59] Walshaw, C., *The Jostle User Manual: Version 2.2*, University of Greenwich, London, 2000.

[60] Karypis, G. and Kumar, V., *METIS Manual, Version 4.0*, University of Minnesota, Department of Computer Science, 1998.

[61] Pellegrini, F., SCOTCH 3.1 User's Guide, TR 1137-96, LaBRI, University of Bordeaux, 1996.

[62] Karypis, G. and Kumar, V., A coarse-grain parallel formulation of multilevel k-way graph-partitioning algorithm, *Proc. SIAM Conf. on Parallel Processing for Scientific Comp.*, Minneapolis, Minnesota, USA, 1997.

[63] Walshaw, C., Cross, M., and Everett, M. G., Parallel dynamic graph partitioning for adaptive unstructured meshes, *J. Parallel Dist. Comp.*, 47(2), 102, 1997.

[64] Walshaw, C. and Cross, M., Mesh partitioning: A multilevel balancing and refinement algorithm, *SIAM J. Scientific Comp.*, 22(1), 63, 2000.

[65] Elsner, U., Static and Dynamic Graph Partitioning. A Comparative Study of Existing Algorithms, Ph.D. Thesis, Technical University Chemnitz, 2002.

[66] Schamberger, S., Improvements to the helpful-set heuristic and a new evaluation scheme for graphs-partitioners, *Intl. Conf. on Computational Science and its Appl.,* Lecture Notes in Computer Science, Vol. 2667, Springer, Berlin, 2003, p. 49.

[67] Hendrickson, B., Graph partitioning and parallel solvers: has the emperor no clothes? in *Solving Irregulary Structured Problems in Parallel*, Ferreira, A., Rolim, J., Simon, H., and Teng, S.-H., Lecture Notes in Computer Science, Vol. 1457, Springer, Berlin, 1998, p. 218.

61

Hypergraph Partitioning and Clustering

David A. Papa
University of Michigan

Igor L. Markov
University of Michigan

61.1 Introduction

A hypergraph is a generalization of a graph wherein edges can connect more than two vertices and are called hyperedges. Just as graphs naturally represent many kinds of information in mathematical and computer science problems, hypergraphs also arise naturally in important practical problems, including circuit layout, Boolean SATisfiability, numerical linear algebra, etc. Given a hypergraph H, *k-way partitioning* of H assigns vertices of H to k disjoint nonempty partitions. The k-way partitioning problem seeks to minimize a given cost function of such an assignment. A standard cost function is *net cut*, which is the number of hyperedges that span more than one partition, or, more generally, the sum of weights of such edges. Constraints are typically imposed on the solution, and make the problem difficult. For example, certain vertices can be fixed in their partitions (fixed constraints) or the total vertex weight in each partition may be limited (balance constraints). With balance constraints, the problem of optimally partitioning a hypergraph is known to be NP-hard [1]. However, since partitioning is critical in several practical applications, heuristic algorithms were developed with near-linear runtime. Such move-based heuristics for k-way hypergraph partitioning appear in Refs. [2–4], with refinements given by Refs. [5–14]. The following is an introduction to partitioning formulations and algorithms, centered on the Fiduccia–Mattheyses (FM) heuristic [3] and its derivatives.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

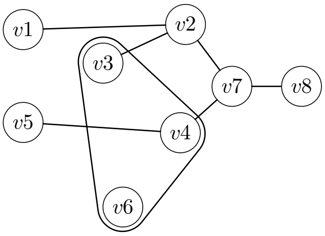


FIGURE 61.1 An example of a nearly block-diagonal matrix and corresponding hypergraph. Each row of the matrix corresponds to a hyperedge, and each column corresponds to vertices $v1$ through $v8$. Recursively bisecting the graph aligns the blocks of the matrix on the diagonal.

There is a wide variety of contexts for hypergraph partitioning. Several of them are outlined in Section 61.2. Each context uses a hypergraph to represent another kind of data structure. The mappings of several mathematical structures to hypergraphs are described below:

Matrices. The pattern of nonzero entries of a matrix **A** can be represented by a hypergraph whose hyperedges correspond to rows of **A** and vertices correspond to the columns of **A**. Each hyperedge, e, will be connected to a vertex, v, if $\mathbf{A}_{e,v} \neq 0$. Figure 61.1 gives an example of a matrix and its corresponding hypergraph.

Logic circuits. Logic circuits are composed of gates (or standard cells) that perform logical operations and connected by metal wires. The same electrical signal may propagate from one gate to several other gates—such a connection is called a net, and can be conveniently represented by a hyperedge. The hypergraph corresponding to a logic circuit directly maps gates to vertices and nets to hyperedges. The dual of this hypergraph is sometimes used as well. In the dual hypergraph, vertices correspond to nets, and hyperedges correspond to gates. An example of a logic circuit and corresponding hypergraph is given in Figure 61.2.

Boolean formulae. The *conjunctive normal form* (CNF) for Boolean formulae consists of Boolean variables or their complements grouped into *clauses* and combined with the OR operation. All of the clauses are then combined with the AND operation (see Figure 61.3 for an example). To convert a CNF formula into a hypergraph, the following mapping is used. Each *literal* in the formula (a Boolean variable is a

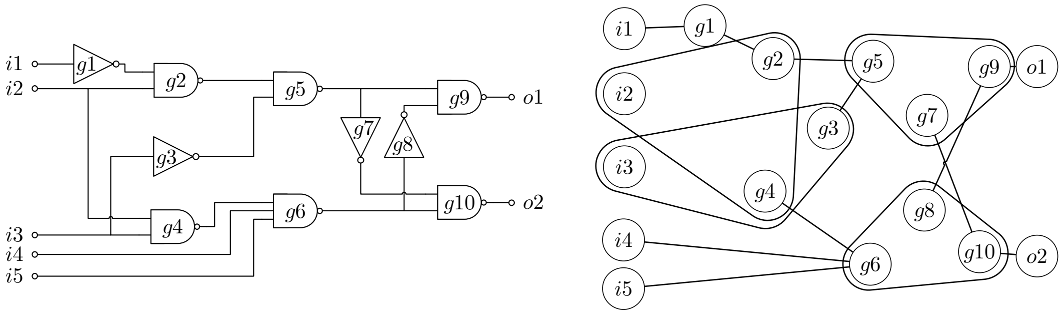


FIGURE 61.2 An example of a logic circuit and the corresponding hypergraph.

$$\phi = (v1 + v3)(v3 + v5)(v2 + v5 + v4)$$
$$(v5 + v6)(v6 + v7 + v8)(v8 + v9)$$

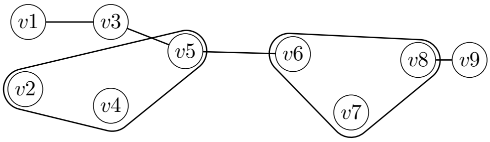


FIGURE 61.3 An example of a CNF logic formula ϕ and the corresponding hypergraph.

literal and its complement is another literal) maps to one vertex in the hypergraph. Each clause maps to a hyperedge, which connects to the vertices that correspond to the literals in the clause. A Boolean formula in conjunctive normal form is shown in Figure 61.3 along with the corresponding hypergraph.

The remainder of this survey discusses hypergraph partitioning as illustrated by these three contexts. Section 61.2 describes how these contexts give rise to practical applications of hypergraph partitioning. Terms used in this survey are defined in Section 61.3. Section 61.4 summarizes the various techniques presented in this survey and outlines how scale dictates which technique is most applicable. The FM heuristic is described in detail in Section 61.5 and the multilevel FM extension is discussed in Section 61.6. Available software and benchmarks are briefly described in Section 61.7. Sample empirical results are given in the appendix.

61.2 Motivation and Applications

Hypergraph partitioning arises in several practical applications, three of which are outlined below.

61.2.1 Numerical Linear Algebra

The runtime of linear algebra computations can vary dramatically depending on the sparsity of input matrices and their patterns of non-zero values [15,16], which can affect the sparsity of intermediate matrices appearing during computations. In particular, many linear algebra operations (such as matrix-vector multiply, matrix-matrix multiply, solving systems of linear equations, eigenvalue problems, etc.) are faster for block-diagonal matrices due to parallelization.

To leverage this speedup, one would like to reorder rows of the matrix to bring non-zero elements in input matrices as close to the diagonal as possible before applying such operations. Fortunately, the results of the operations mentioned above are not changed substantially by row or column permutation. Finding an optimal permutation is computationally hard [59], but one can use recursive calls to partitioning on a hypergraph representation of the matrix to find an appropriate solution. Figure 61.1 shows a small example of a sparse block-diagonal matrix whose rows were ordered in this fashion.

61.2.2 Integrated Circuit Design

Very Large Scale Integration (VLSI) circuit design has long provided driving applications and ideas for hypergraph partitioning heuristics. For example, the methods of Kernighan–Lin [2] and Fiduccia–Mattheyses [3] form the basis of today's move-based approaches. The method of Goldberg–Burstein [17] presaged the multilevel approaches recently popularized in the parallel simulation [18–20] and VLSI [10,13,21] communities. As noted in Ref. [22], applications in VLSI design include test, simulation and emulation; design of systems with multiple field-programmable devices; technology migration and repackaging; and top–down floorplanning and placement.

Depending on the specific VLSI design application, a partitioning instance may have directed or undirected hyperedges, weighted or unweighted vertices, and so on. However, in all contexts the instance represents at the transistor, gate, cell, block, chip, or behavioral description module level—a human-designed system. Such instances are highly nonrandom. Hence, the current practice remains to evaluate new algorithmic ideas against suites of human-designed benchmark instances. In the VLSI partitioning community, performance of algorithms is typically evaluated on the ISPD 1998 benchmarks released by IBM. Alpert [23] noted that previous circuits did not reflect the complexity of modern partitioning instances, particularly in VLSI physical design; this motivated the release of 18 larger benchmarks produced from internal designs at IBM [24].

Salient features of benchmark (real-world) circuit hypergraphs include:

- *Size.* Number of vertices can be in the millions (instances of all sizes are equally important).
- *Sparsity.* Average vertex degrees are typically between 3 and 5 for device-, gate-, and cell-level instances; higher average vertex degrees occur in block-level design.

- Number of hyperedges (nets) typically between $0.8x$ and $1.5x$ of the number of vertices (each module typically has only one or two outputs, each of which represents the source of a new signal net).
- Average net sizes are typically between 3 and 5.
- A small number of very large nets (e.g., clock, reset, test) connect hundreds or thousands of vertices.

Partitioning heuristics must be highly efficient to be useful in VLSI design.[1] Because of this and also because of their flexibility in addressing variant objective functions, fast and high-quality iterative move-based partitioners using the approach of Fiduccia–Mattheyses [3] have dominated recent practice.

The primary use of partitioning heuristics in VLSI design is that of top–down recursive min-cut bisection placement. In this placement framework, a region of a chip is divided geometrically, and the logic inside that region is partitioned topologically. Each of these pieces are then recursively divided until the regions are so small that an optimal end-case placer can solve the problem in a reasonable amount of time.

61.2.3 Automated Theorem-Proving and Formal Verification

Algorithms for electronic design automation (EDA) [25,26], including those for synthesis as well as hardware and software verification, require efficient manipulation of Boolean functions. Boolean satisfiability (SAT) [27,28] solvers and binary decision diagrams (BDDs) [29] have traditionally been used with such applications, but their worst-case complexity remains exponential and can scarcely be improved.

A key observation is that Boolean functions arising in EDA applications and constraint satisfaction problems possess useful structural properties, for example, related variables in satisfiability typically participate in the same clauses. Uses of problem structure are known to improve the efficiency of SAT and BDD algorithms. For example, Prasad et al. [30] theoretically show that combinational circuits with small net cuts give easy instances of automatic test pattern generation (ATPG), which are essentially SAT instances. BDDs with smaller cuts tend to have fewer hyperedges and vertices, speeding up BDD manipulations [31,32].

Based on these observations, Aloul et al. [33] reorder Boolean variables to place "connected" variables close to each other. The ordering process relies on recursive calls to hypergraph partitioning to reduce net cut. This optimization can accelerate SAT solving and BDD manipulation, and reduce BDD memory consumption. The authors of Refs. [34,35] apply partitioning techniques to more general types of reasoning and more sophisticated theorem provers.

61.3 Definitions and Terminology

In what follows, V is the set of vertices in a hypergraph.

Definition 61.1 **(Disjoint Partitions)**

A k-tuple $\mathbf{P} = (\mathbf{p}_0, \ldots, \mathbf{p}_{k-1})$ *with each* $\mathbf{p}_i$ *a set of vertices such that* $\bigcup_{i=0}^{k-1} \mathbf{p}_i = \mathbf{V}$ *and* $\bigcap_{i=0}^{k-1} \mathbf{p}_i = \emptyset$.

Definition 61.2 **(k-Way Partitionment)**

A function of the form $\delta : \mathbf{V} \to \mathbf{P}$ *wherein all vertices of* $\mathbf{V}$ *are mapped to a disjoint partitions from the k-tuple* $\mathbf{P}$. *More practically, this function assigns the vertices to one of k disjoint partitions.*

Definition 61.3 **(Balance Constraint)**

A pair of values (l, u). *A partition* $\mathbf{p}$ *with a balance constraint must obey* $l \leq \sum_{v \in \mathbf{p}} W(v) \leq u$, *where* $W(v)$ *is the weight of vertex v or 1 if the vertex has no weight.*

[1] For example, a modern top–down standard-cell placement tool will perform recursive min-cut bisection of a gate-level hypergraph to obtain a coarse global placement, which is then refined into a detailed placement by local optimizations. This entire placement process, for example, takes approximately 1 CPU minute per 7000 standard cells on an AMD Opteron 250 workstation with adequate RAM. The implied partitioning runtimes are on the order of 1 CPU second for hypergraphs with 3500 vertices, and 10 min for hypergraphs with 2 million vertices.

When $\forall v \in V$, $W(v) = 1$, then $\sum_{v \in \mathbf{p}} W(v) = |\mathbf{p}|$, the number of vertices in $\mathbf{p}$. Partitioning using similar balance constraints for all partitions will equalize the number of vertices in each.

Definition 61.4 (Hypergraph)

A pair of sets $\mathbf{H} = (\mathbf{V}, \mathbf{E})$. $\mathbf{V}$ *is the set of* vertices *of the hypergraph and* $\mathbf{E}$ *is the set of* hyperedges *of the hypergraph. Each hyperedge in a hypergraph is a nonempty subset of* $\mathbf{V}$*, the size of this subset is called the hyperedge's* degree. *A weighted hypergraph has nonnegative numeric weights associated with each vertex, each hyperedge, or both.*

Conventional graphs are a special case of hypergraphs, where all hyperedges have degree 2. In most cases, the degree of a hyperedge is no smaller than 2.

Vertices and hyperedges optionally have nonnegative numeric weights. A weight of 0 usually means that the edge or the vertex is deleted. Weights usually have additive semantics, for example, two weighted edges, e_1 and e_2, connecting the same pair of vertices can be replaced with a single edge, e_3, such that $W(e_3) = W(e_1) + W(e_2)$ (see also Definition 61.8).

Definition 61.5 (Cut)

A hyperedge $\mathbf{e}$ *of hypergraph is cut if, with respect to a particular partitionment* δ*, its vertices are mapped to more than one partition. The net cut of a partitionment is the total number of hyperedges that are cut. The* weighted net cut *of a partitionment is the sum of the weights of hyperedges that are cut.*

Definition 61.6 (Hypergraph Partitioning)

The process of finding a partitionment of a hypergraph such that some cost function, such as net cut, is minimized. When the solutions must additionally satisfy balance constraints, the process is called balanced hypergraph partitioning. Hypergraph partitioning that results in two partitions is called bisection.

Definition 61.7 (The *k*-Way Hypergraph Partitioning Problem)

Given a hypergraph $\mathbf{H} = (\mathbf{V}, \mathbf{E})$*, find a k-way partitionment* $\delta : \mathbf{V} \to \mathbf{P}$ *that maps the vertices of* $\mathbf{H}$ *to one of k disjoint partitions such that some cost function* $c : \delta \to \mathbb{R}$ *is minimized.*

One typically deals with a particular hypergraph partitioning problem *instance* which consists of a particular hypergraph, a set of two or more balance constraints, and an objective function.

Definition 61.8 (Clustering)

The process of computing a coarser hypergraph from an input hypergraph by merging vertices into larger groups of vertices called clusters.

The weight of each cluster will be the sum of the weights of its vertices, or simply the number of vertices if they have no weights.

Several cost objectives exist for the hypergraph partitioning problem. The most common by far is net cut. In VLSI placement, reduced net cut is correlated with shorter wires, and in parallelization smaller net cut means reduced interprocessor communication. For more than 2-way partitioning, the sum-of-degrees metric is sometimes used. For this metric, the cost of each hyperedge is equal to the number of different partitions which contain some of its vertices.

61.4 Summary of Techniques and Their Scale Dependence

In many applications of hypergraph partitioning, the size of input grows every year, and the demand for performance is high. For example, the number of transistors in a typical VLSI design continues to grow exponentially (according to Moore's Law). The algorithms applicable to these circuits must scale to tens of millions of components today and hundreds of millions in the foreseeable future. Because of these large inputs, any partitioning technique used must have near-linear complexity in the worst case to be effective.

State-of-the-art partitioning tools use local search heuristics to refine a given partitionment. The basic technique common to all VLSI partitioning applications is the FM algorithm [3], which applies linear-time *passes* to iteratively improve a given partitionment by moving every vertex exactly once. A prevalent extension of this algorithm is the MLFM algorithm, which improves both solution quality and runtime of partitioning large hypergraphs by *clustering* tightly connected components and partitioning the resulting smaller hypergraph. However, for sufficiently small hypergraphs, "flat" FM can produce optimal solutions faster than MLFM, indicating that different techniques should be applied at different scales.

Despite growing input sizes, recursive applications of partitioning will still produce a large number of small partitioning instances, and the performance of partitioning on small instances is equally relevant in practice. In high-performance applications, actual runtime, rather than asymptotic complexity, is a design objective of prime importance. Due to this, occasionally an algorithm with higher complexity will be selected if it is faster for practical input sizes. The following is a summary of the techniques available, and at which scale they are effective.

61.4.1 Exhaustive Enumeration

Exhaustive enumeration produces optimal partitionments and has exponential asymptotic complexity but low constant runtime factor. As such, at very small scales, exhaustive enumeration is the fastest known technique. The evaluation of each solution is the bottleneck of this algorithm, and it can be sped up by incrementally evaluating the cost objective. However, it is straightforward to iteratively update the cut of a partitionment when one vertex is moved [36]. We can represent each partitionment as a vector, with each entry corresponding to a vertex's partition. Then, enumerating the values of this vector in Gray code order moves only one vertex at a time, thus allowing incremental update of net cut for fast evaluation. Empirically, we find this to be the fastest technique for one to nine vertices [36].

61.4.2 Branch and Bound

The runtime of exhaustive enumeration grows exponentially in the number of vertices and thus cannot scale very far. However, its scalability can be improved through intelligent search space pruning; this technique is known as branch and bound (B&B). A B&B implementation does a depth-first search of the tree of partial partitionments (some vertices assigned to a partition, some yet unassigned) by choosing a partition for the next unassigned vertex and recursing. It can search the entire solution space in the worst case and therefore also has exponential time complexity, but maintains optimality by bounding away only suboptimal results. Partition balance constraints can be used for pruning illegal solutions and best-seen cost can be used to bound away suboptimal results.

One notable improvement to the bounding function is known as *inevitable cut* [36], which applies to the portion of a hypergraph that is made up of two-pin nets. Choosing *any* partition for a particular vertex may imply some additional cost, because it may be connected to vertices in both (all) partitions. The minimum possible additional cost (considering only two-pin nets) over all legal partition assignments for a particular vertex is known as its *inevitable cut*, and can be computed directly. It is the minimum number of two-pin connections to vertices locked in any particular partition. Figure 61.4 shows two small examples

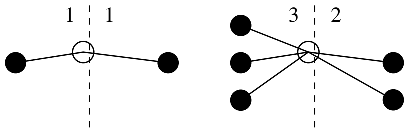


FIGURE 61.4 Two small examples of an inevitable cut computation. The black vertices have been assigned to their respective partitions and the white vertex is still unassigned. In each case, any assignment of the white vertex implies that some edges will be cut. The inevitable cut is 1 on the left and 2 on the right. For a given unassigned vertex, inevitable cut is computed as the smallest number of adjacent vertices assigned to any one partition.

of an inevitable cut computation (for more details see Ref. [36]). The inevitable cut of a vertex can be safely added to the lower-bound cost computation in the bounding function to strengthen the pruning of suboptimal results. It is unclear how to extend this technique to hyperedges of degree greater than 3. The inevitable cut technique can be extended to handle a larger portion of the hypergraph by replacing degree-3 hyperedges with three-cliques (triangles), with each hyperedge's weight multiplied by one-half the weight of the original degree-3 hyperedge. This "triangle technique" preserves cost of all partitionments exactly and allows inevitable cut to be applied to all degree-2 and degree-3 hyperedges. The runtime of B&B can be unreasonably high for certain pathological cases that have many optimal solutions. In practice we detect these cases with a time-out. If the runtime limit is exceeded we will stop B&B and use a more scalable algorithm such as FM discussed below. Empirically, we find B&B to be the best technique for 10–35 movable objects [36].

61.4.3 The Fiduccia–Mattheyses Heuristic

Partitioning problems with more than 35 movable objects take an impractical amount of runtime to solve optimally using B&B. Fortunately, an amortized near-linear-time heuristic exists for iterative improvement of hypergraph partitions. The FM algorithm works by prioritizing *moves* by *gain*. A *move* changes to which partition a particular vertex belongs, and the *gain* is the corresponding change to the cost function. After each vertex is moved, gains for connected modules are updated.

The FM algorithm runs in *passes* wherein each vertex is moved exactly once. Passes are generally applied until little or no improvement remains. Initial solutions are often produced using a simple randomized algorithm. Empirically, we find FM to be the best technique for 36–200 movable objects. Details of the FM algorithm are discussed below and, at length in Section 61.5.

61.4.4 Multilevel Fiduccia–Mattheyses Framework

The multilevel hypergraph-partitioning framework provides the best known partitioning results for large-scale circuit hypergraphs. It consists of three main components: clustering, top-level partitioning, and refinement. During clustering, hypergraph vertices are combined into clusters based on connectivity, leading to a smaller, clustered hypergraph. This step is repeated until the hypergraph is small enough to be solved effectively by a "flat" partitioning algorithm (e.g., the FM heuristic). The smallest hypergraph is partitioned with a very fast initial solution generator (e.g., random) and iteratively improved using the flat partitioner. The resulting partitionment is then interpreted as a solution for the next (less clustered) hypergraph during the refinement stage, and improved again using the flat partitioning algorithm until the bottom level is reached.

Using this multilevel framework with the FM algorithm is known as the MLFM technique, and it is the most effective and commonly used hypergraph partitioning algorithm for large circuit hypergraphs. Empirically, we find the MLFM algorithm to be the best technique for more than 200 movable objects. Details of the MLFM algorithm are discussed at length in Section 61.6.

61.4.5 Other Types of Algorithms

Several other approaches exist for solving the hypergraph partitioning problem [37–41]. Typically, these techniques have some substantial drawback that makes their use impractical for high-performance applications. For example, metaheuristics such as simulated annealing and tabu search may be able to achieve better solution quality at the cost of an impractical increase in runtime [39]. Other techniques may have constraints on their input that are violated by some problem instances, as is the case with spectral techniques [38]. Spectral algorithms find eigenvalues of the Laplacian matrix of the connectivity graph and derive a partitionment from coefficients of an eigenvalue, for example, by comparing them to the median [42]. These algorithms may not handle fixed terminals well and are therefore not general enough to handle many practical applications. Another technique relies on the Min-cut Max-flow theorem and efficient network-flow algorithms to identify optimally small cuts in graphs [41]. These polynomial-time algorithms do not typically perform approximation and can handle hyperedges using accurate conversion to graphs, but do not handle balance constraints, which results in expensive trial-and-error in flow-based partitioners.

61.5 The Fiduccia–Mattheyses Heuristic

The FM heuristic for partitioning hypergraphs [3] is a linear-time iterative improvement algorithm. Its neighborhood structure is induced by single-vertex, partition-to-partition moves.[2] FM starts with a possibly random solution and changes the solution by a sequence of moves which are organized as *passes*. At the beginning of a pass, all vertices are free to move (*unlocked*), and each possible move is labeled with the immediate change to the cost it would cause; this is called the gain of the move (positive gains reduce solution cost, while negative gains increase it). Iteratively, a move with highest gain is selected and executed, and the moving vertex is locked, that is, it is not allowed to move again during that pass. Since moving a vertex can change gains of adjacent vertices, after a move is executed all affected gains are updated. Selection and execution of a best-gain move, followed by gain update, are repeated until every vertex is locked. Then, the best solution seen during the pass is adopted as the starting solution of the next pass. The algorithm terminates when a pass fails to improve solution quality. Pseudocode for the FM heuristic is given in Figure 61.5.

The FM algorithm has three main components: (1) the computation of initial gain values at the beginning of a pass; (2) the retrieval of the best-gain (feasible) move; and (3) the update of all affected gain values after a move is made. One contribution of Fiduccia and Mattheyses lies in observing that circuit hypergraphs are sparse, and any move's gain is bounded between plus and minus the maximal vertex degree in the hypergraph (times the maximal hyperedge weight, if weights are used). This allows prioritization of moves by their gains. All affected gains can be updated in amortized-constant time, giving overall linear complexity per pass [3]. In Ref. [3] all moves with the same gain are stored in a linked list representing a "gain bucket." It is important to note that some gains may be negative, and as such, FM performs hill climbing and is not purely greedy.

```
FM(hypergraph, partitionment)
     do
         initialize gain_container from partitionment;
         FMpass(gain_container, partitionment);
     while(solution quality improves);
FMpass (gain_container, partitionment)
     solution_cost = partitionment.get_cost();
     while(not all vertices locked)
         move = choose_move(gain_container);
         solution_cost += gain_container.get_gain(move);
         gain_container.lock_vertex( move.vertex() );
         gain_update(move, gain_container);
         partitionment.apply(move);
     roll back partitionment to best seen solution;
     gain_container.unlock_all();
```

FIGURE 61.5 Pseudocode for the FM heuristic. choose_move and gain_update are defined in Figure 61.6.

[2] By contrast, the stronger Kernighan–Lin (KL) heuristic [2] uses a pair-swap neighborhood structure and has cubic runtime per pass.

61.5.1 Fiduccia–Mattheyses Passes

The FM algorithm consists of incrementally improving passes [3]. During each pass, FM will search the neighborhood of a given partitionment, and record the best seen solution. FM performs successive passes as long as the solution can be improved.

At each pass, the FM repetitively chooses one (best) move and applies it, followed by the processing of information about the new solutions thus obtained. Since no vertex can be moved twice in a pass, no moves will be available beyond a certain point (*end of a pass*). Some best-gain moves may increase the solution cost, and typically the solution at the end of the pass is not as good as the best solutions seen during the pass. FM will then undo a given number of moves to return to a solution with best-seen cost.

61.5.2 Gain Computation and Gain Update

The initialization of the FM data structures at the beginning of each pass is straightforward. First, traverse all hyperedges and count the number of vertices in each partition. If a hyperedge is uncut, then decrease the gain of all adjacent vertices by the hyperedge weight (or by 1 when no weights are given). If there is only one vertex in some partition, then increase the gain of that vertex by the hyperedge weight (or by 1 when no weights are given) and do not change the gain of all other adjacent vertices.

Picking and applying one move is subtle. FM requests the best move from the gain container and can continue requesting more moves until a feasible (i.e., not violating the balance constraints) move is found. As FM applies the chosen move and locks the vertex, gains of adjacent vertices likely need to be updated. In performing "generic" gain update, an implementation of FM must walk all hyperedges incident to the moving vertex and for each hyperedge computes gain updates (*delta gains*) for each of its vertices because of this hyperedge (these are combinations of the given hyperedge's cost under four distinct partition assignments for the moving and affected vertices; see Figures 61.6 and 61.7). These partial gain updates are immediately applied to the gain container, and moves of affected vertices may have their priority within the gain container changed. Even if the delta gain for a given move is zero, removing and inserting it into the gain container will typically change tie-breaking among moves with the same gain.

```
choose_move (gain_container)
     move = gain_container.max_feasible_move();
     while(move is infeasible)
         gain_container.mark_infeasible(move);
         move = gain_container.max_feasible_move();
     gain_container.mark_all_feasible();
     return move;
```

```
gain_update (move, gain_container)
     source_part = partition that move.vertex() is in;
     dest_part = partition where move.vertex() is going;
     for_each(hyperedge e incident to move.vertex())
         if(e is not cut before applying move)
             for_each(vertex v on e)
                 for_each(partition p != source_part)
                     gain_container.update(v, p, e.weight());
         if(e is cut before applying move and uncut after)
             for_each(vertex v on e)
                 for_each(partition p != dest_part)
                     gain_container.update(v, p, -e.weight());
         if(only 1 vertex v remains outside dest_part after applying move)
             gain_container.update(v, dest_part, e.weight());
         if(only 1 vertex v outside source_part before applying move)
             gain_container.update(v, source_part, -e.weight());
```

FIGURE 61.6 Functions called by the FM heuristic. A faster version of gain_update that takes advantage of special cases and bipartitioning is given in Figure 61.7.

```
gain_update_special_cases ( move )
    source_part = partition that move.vertex() is in;
    dest_part = partition where move.vertex() is going;
    for_each(hyperedge e incident to move.vertex() )
        if( e.degree() == 2)
            v = the vertex on e that is not move.vertex();
            if( v is not locked and v is in source_part)
                gain_container.update( v, dest_part, 2*e.weight() );
            else if( v is not locked)
                gain_container.update( v, source_part, -2*e.weight() );
        else if( tallies[dest_part] == 0)
            for_each( vertex v on e )
                if( not locked( v )
                    gain_container.update( v, dest_part, e.weight());
        else if( tallies[source_part] == 1)
            for_each( vertex v on e )
                if( not locked( v )
                    gain_container.update( v, source_part, -e.weight());
        else
            for_each( vertex v on e )
                if( not locked( v ) )
                    if( v is in source_part )
                        if( tallies[source_part] == 2 )
                            gain_container.update( v, dest_part, e.weight() )
                            break;
                    else if( tallies[dest_part] == 1)
                        gain_container.update( v, source_part, -e.weight() )
                        break;
```

FIGURE 61.7 Pseudo-code for a faster gain update that takes advantage of special cases and bipartitioning.

In most implementations the gain update is the main bottleneck, followed by the gain container construction. The net cut objective is particularly amenable to optimizations during gain update [3,6,44]. For example, hyperedges that have at least one vertex locked in each partition can be safely ignored during gain update, as the cost of such a hyperedge cannot change until the end of a pass, and they do not contribute to gain update. We term such hyperedges *locked.* Once a hyperedge becomes locked, it is marked as such in a dedicated bitvector. While this requires additional effort, the savings in the overall runtime per pass are considerable.

61.5.3 Custom Data Structures

The efficient custom data structures used by the FM algorithm are critical to its performance. The *gain bucket list* data structure introduced in Ref. [3] is necessary to allow linear-time gain update. Figure 61.8 shows the gain bucket list structure as originally illustrated. On the left-hand side is a *bucket-array* structure which stores $2 * pmax + 1$ pointers to gain buckets, where $pmax$ is the maximum possible gain. The bucket-array allows constant-time lookup of moves which have a particular gain. Each bucket is a possibly empty doubly linked list of *gain elements.* On the bottom is a *repository*, which stores pointers to the gain elements associated with each vertex. The repository allows constant time lookup of the gain for a particular vertex. To efficiently find the move with max-gain, the index of the highest gain, nonempty bucket is maintained.

The bucket-array structure is efficiently implemented with an array when the magnitude of hyperedge weights is limited by some constant. However, an array-based implementation of the bucket-array has space complexity dependent upon the magnitude of hyperedge weights. For arbitrary hyperedge weights, the bucket-array must be modified to efficiently support the operations of a bucket-array in the context

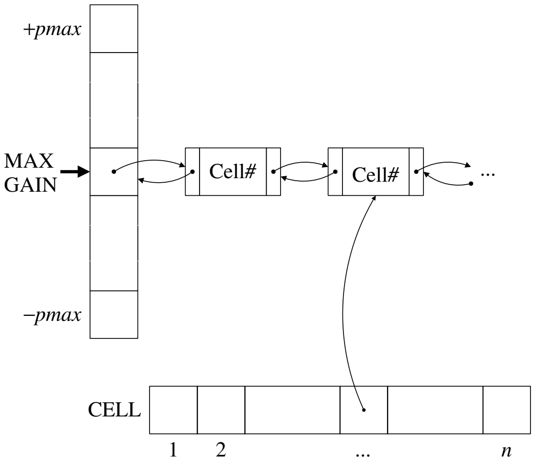


FIGURE 61.8 The gain bucket list structure as illustrated in Ref. [3]. More efficient implementations than what is depicted exist.

of an FM gain container. These operations include (1) finding a particular gain bucket; (2) insertion and removal of gain elements; (3) finding the maximum nonempty gain bucket; and (4) finding the second-highest nonempty gain bucket. The last operation is necessary, for example, when the max-gain bucket is exhausted and the new max-gain must be determined. If hyperedge weights are large, then an array-based implementation of bucket-array will result in a large, sparse, and wasteful structure that must be searched (linear in the magnitude of weights) to implement operation (4).

Figure 61.9 shows an alternative implementation of bucket-array that allows for time complexities of operations (1), (3), and (4) that do not depend on the magnitude or sparsity of hyperedge weights. Operation (2) is logarithmic in the number of nonempty gain buckets. On the right-hand side is a hash table that stores pointers to gain buckets. Since the expected lookup time for a hash table is constant, this

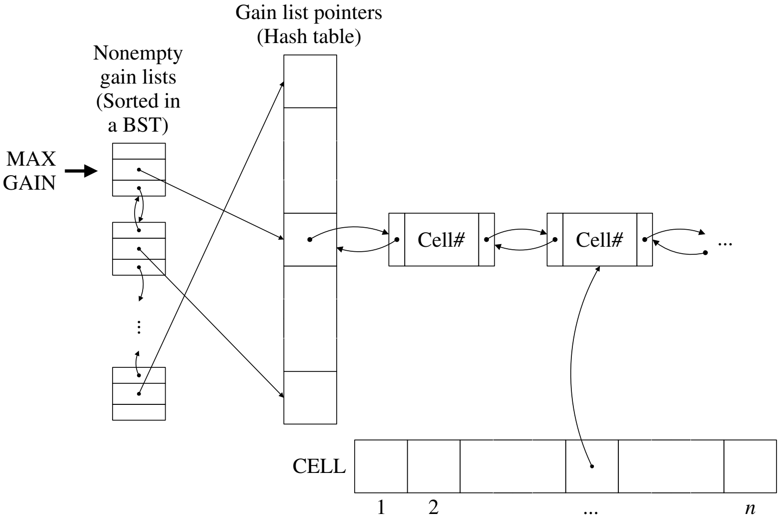


FIGURE 61.9 The gain list structure with a hybrid BST+hash-table bucket-array for improved FM complexity with arbitrary hyperedge weights.

efficiently implements operation (1). On the left-hand side is a binary search tree (BST)[3] that stores the gains which are possible from some move (i.e., nonempty gain lists). Since a BST is sorted, the largest item in the BST can be found in constant time, and this is an efficient implementation of operation (3). Operation (4) is as easy as operation (3) for this structure, since the second-highest gain is the second from the last entry in the nonempty gain BST. The most difficult operation for this structure is operation (2), since BST insertion and removal are logarithmic time operations. Nevertheless, the complexity of operation (4) for an array-based implementation is linear in the magnitude of hyperedge weights. Therefore, for sufficiently large weights, the runtime of this BST+hash bucket-array will be less than that of an array-based implementation of bucket-array.

61.5.4 Implementation Insights

There are important degrees of freedom in the implementation of the above gain containers which can have a significant effect on solution quality. Among the most notable is whether items are taken from the gain buckets in Last-In-First-Out (LIFO) or First-In-First-Out (FIFO) order. It has been shown that choosing LIFO order gain buckets can provide significant improvement in solution quality [7]. The intuition for this effect is that it exploits some locality in the structure of the hypergraph. When moving one vertex to a different partition causes a change to the gain of another vertex, that other vertex should also be considered for movement.

Extending the theme of exploiting the locality in graphs, Dutt and Deng [9] propose the CLuster-oriented Iterative-improvement Partitioner (CLIP) algorithm. The key observation in this algorithm is that if tightly connected cluster is cut, then the solution cost can be reduced by moving the cluster entirely into a single partition. CLIP first computes the gains of all vertices. It then sorts them into a single linear order and puts them all into the zero-gain bucket. This allows for choosing the highest gains first. More importantly, as partitioning proceeds, the gain of each move will be determined solely by the change in gain due to moves of adjacent vertices. Thus, when part of a cluster is moved, it is more likely that the rest of the cluster will be moved along with it. CLIP does not track net cut as accurately as conventional FM, and therefore it usually does not improve initial solutions that are already good. CLIP is used in nonincremental contexts, starting with a random initial solution and is typically postprocessed by conventional FM passes.

FM also tends to have problems when the highest-gain move is a vertex with high weight that cannot move across the cutline because of balance constraints. This is known as the *corking effect* and some techniques for handling it are presented in Ref. [45]. Techniques specific to hypergraphs with fixed vertices are presented in Ref. [46].

61.6 Multilevel Fiduccia–Mattheyses Partitioning Framework

The multilevel hypergraph partitioning framework was successfully verified in 1997 by Alpert et al. [10], Karypis et al. [13], and Karypis and Kumar [47] and has been conducive to the best known partitioning results ever since. The main advantage that MLFM has over flat partitioners is its ability to more effectively search the solution space by spending comparatively more effort on smaller coarsened hypergraphs. Good coarsening algorithms allow for high correlation between good partitionments for coarsened hypergraphs and good partitionments for the initial hypergraph. Therefore, a thorough search at the top of the multilevel hierarchy is worthwhile because doing so is relatively inexpensive when compared to flat partitioning of the original hypergraph, but can still preserve most of the possible improvement. The result is an algorithmic framework with both improved runtime and solution quality over a completely flat approach. Pseudocode for an implementation of the MLFM framework is given in Figure 61.10.

[3] We assume an implementation of a BST that allows constant time traversal of the sorted sequence. The standard C++ set is an example of such a data structure.

```
MLFM(hypergraph)
    level = 0;
    hierarchy[level] = hypergraph;

//Coarsening phase
    while(hierarchy[level].vertex_count() > 200)
        next_level = cluster(hierarchy[level]);
        level = level + 1;
        hierarchy[level] = next_level;

//Top level partitioning phase
    partitionment[level]
            = a random initial partitionment for top-level hypergraph;
    FM(hierarchy[level], partitionment[level])

//Refinement phase
    while(level > 0)
        level = level - 1;
        partitionment[level]
            = project(partitionment[level+1], hierarchy[level]);
        FM(hierarchy[level], partitionment[level])

    return partitionment[0];
```

FIGURE 61.10 Pseudocode for the MLFM algorithm.

Multilevel partitioning consists of three main components: clustering, top-level partitioning, and refinement or "uncoarsening." During clustering, hypergraph vertices are combined into clusters based on connectivity, leading to a smaller, clustered hypergraph. This step is repeated until there are only several hundred clusters, culminating in a hierarchy of clustered hypergraphs. We describe this hierarchy with the smaller hypergraphs being "higher" and the larger hypergraphs being "lower."[4] The smallest (top-level) hypergraph is partitioned with a very fast initial solution generator and iteratively improved, for example, using the FM algorithm. The resulting partitionment is then interpreted as a solution for the next hypergraph in the hierarchy. During the refinement stage, solutions are projected from one level to the next and iteratively improved, for example, by the FM algorithm.

Additionally, the hMETIS partitioning program [48] introduced several new heuristics that are incorporated into their multilevel partitioning implementation and are reportedly performance-critical. One is hyperedge removal during refinement, which is analogous to FM, except that a single move "uncuts" a hyperedge by reassigning as many vertices as needed. Another heuristic is V-cycling, a repetition of the clustering-partitioning-refinement process that uses a solution produced by a previous execution of this process—vertices in different partitions cannot be clustered. A similar technique is v-cycling, in which the refinement stage may stop before the bottom-level hypergraph is reached and clustering resumed (starting from a solution for a clustered hypergraph). Similarly, clustering may be stopped earlier than it would normally be, and refinement resumed.

61.6.1 Coarsening

The multilevel partitioning framework begins by "coarsening" the input hypergraph which results in smaller hypergraphs that retain as much of the structure of the original hypergraph as possible. Solutions to these coarsened hypergraphs are then interpreted as solutions to larger hypergraphs during refinement. "Clustering," the name for merging two or more vertices together, is the mechanism by which hypergraphs

[4] This is the most common notation used for a multilevel partitioning hierarchy. hMETIS related works invert the notation, including the naming of V-cycles which may be more appropriately called Λ-cycles (lambda cycles).

are coarsened. Here we cover the details of two simple but effective clustering algorithms. A countless number of more complex schemes exist in partitioning literature, [11,49–52].

61.6.1.1 Edge Coarsening

A simple linear-time clustering strategy called (hyper)Edge Coarsening (EC) was proposed in Ref. [10,13]. The EC technique works by combining connected vertices and is commonly randomized by choosing a vertex and a random neighbor to merge. A straightforward but effective EC implementation has the following attributes [46]:

- The hypergraph is updated continuously as the clustering occurs, that is, the next pair of merged clusters is selected with the knowledge of the last merged pair.
- No cluster can be merged with another if its weight is more than four to five times the average cluster weight at the current level.
- Hyperedge weights are additionally divided by the square root of the sum of cluster weights to discourage merging large clusters.
- Clustering ratio used is 1:3, unclustering ratio is 1:2.8.
- Clustering stops when the clustered hypergraph has 200 clusters or fewer.

61.6.1.2 Heavy Edge Matching

Heavy (hyper)Edge Matching (HEM) is an alternative clustering strategy which seeks to minimize the net cut of the coarsened hypergraph directly. This is achieved by contracting hyperedges with the highest possible weights. In the hypergraph version of this strategy, a vertex is chosen and it will be probabilistically merged with its "nearest neighbor," defined as that with the largest weight connection. The net cut (and partitioning runtime) of a coarsened hypergraph can also be improved when nets disappear entirely due to all of its connected vertices being merged into one. Since it is more likely that a smaller net will be removed, HEM weights connections via smaller nets more highly. A common weighting scheme W is $W(e) = \frac{1}{degree-1}$ for hyperedge e, but many variations of this are possible. HEM also seeks to minimize the number of remaining nets by favoring neighbors which have multiple connections via several nets. So the weight of a connection (u, v) will be $\sum_{e \in E}(W(e)$ if $u, v \in e$ or 0 otherwise).

61.6.2 Top-Level Partitioning

The FM algorithm is only capable of iteratively improving an initial solution. At the top-level no previous solution is available to improve, therefore top-level partitioning requires initial solution generation. A rather simple generator will suffice, because the FM algorithm will quickly find a nearby local minimum. The following "randomized engineering method" (REM) creates random, legal initial solutions. REM first sorts vertices according to decreasing size, so that it can assign the largest vertices first to satisfy balance constraints and avoid "painting itself into a corner." It assigns vertices to partitions in sorted order using biased random selection ("spinning a roulette wheel" such that each outcome has a prescribed probability). Assignment probabilities are proportional to the hypothetical area remaining before the solution satisfies minimal area requirements, the *area slack*, computed after assigning a given vertex to the various partitions. This keeps area slacks approximately equal, yet provides a good degree of randomness. REM will continue assigning vertices to partitions in this way until all partitions reach the lower bound of their respective balance constraints, at which point it will compute slacks relative to the upper bound.

61.6.3 Refinement

After a partitionment is chosen at a particular level, it is projected onto the less clustered (finer) hypergraph at the next level. A vertex in the finer hypergraph is projected into the same partition as the cluster it belongs to in the coarsened hypergraph. The solution for this hypergraph will have cut identical to that for the more clustered hypergraph. However, it is likely that the solution can be improved with respect to the finer hypergraph. This is done through iterative improvement by calling FM on the partitioning solutions at each level.

61.7 Available Software and Benchmarks

There are two common, freely available academic tools for performing hypergraph partitioning. MLPart [53] is an open source C++ implementation of MLFM hypergraph partitioning geared toward circuit hypergraphs and partitioning-based placement. hMETIS [54] is the hypergraph version of METIS, a multilevel graph partitioning algorithm implemented in C. hMETIS is distributed freely for academic use in the form of a precompiled library and executables.

These two partitioning tools use different file formats to represent hypergraphs. hMETIS uses one text file to represent a hypergraph, while MLPart uses several text files in the Bookshelf format to represent partitioning problems, which includes a hypergraph.[5] Both tools additionally define balance constraints in terms of partition capacity targets, and a tolerance parameter that specifies how far it is allowed to deviate from its target.

61.7.1 hMETIS Benchmark Format

The input hypergraph to hMETIS has a simpler format than Bookshelf. The first line contains the number of hyperedges, the number of vertices, and an optional format bit-flag that indicates the presence of weights for hyperedges, vertices, or both. If the vertices have weights, the file will have $|E| + |V| + 1$ lines, and $|E| + 1$ lines otherwise. The file may additionally contain commented lines preceded by %. Each of the following $|E|$ lines represents one hyperedge, and will optionally contain an integer weight, followed by a list of the indices of vertices on that hyperedge. If vertex weights are present, the remaining $|V|$ lines will contain the integer weight of each vertex. The solution is specified by giving the partition of the ith vertex on the ith line of the solution file. Partitioning tolerance is specified with a parameter called *UBfactor* that can be specified on the command line or when making calls to the hMETIS library. This number specifies how far any partition's total weight can deviate from the average weight of a partition, as a percentage. For more details and examples, see Figure 61.11 and Ref. [48].

Line no.	Graph file	Solution file
1	7 8	1
2	1 2	1
3	5 4	0
4	3 4 6	0
5	2 3	0
6	4 7	0
7	2 7	1
8	7 8	1

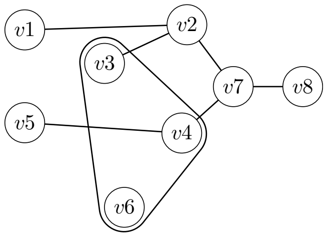


FIGURE 61.11 An example of an hMETIS graph file and partitioning solution for the hypergraph on the right-hand side. Line 1 of the graph file specifies that there are seven hyperedges and eight vertices. The remaining lines specify hyperedges. The ith line of the solution file specifies the partition of the ith vertex. Due to space limitations the Bookshelf version of this example has been omitted but is available online [57].

[5] In addition, industrial VLSI design tools commonly use LEF/DEF files to represent hypergraphs. Tools exist to convert LEF/DEF into the Bookshelf format. See Refs. [55,56] for more information.

61.7.2 Bookshelf Benchmark Format

The Bookshelf benchmark format is more expressive because it is used in other VLSI applications than hypergraph partitioning, including placement and floorplanning. Each Bookshelf partitioning benchmark will be represented with several files, at minimum: a .nodes file containing a list of vertices and their sizes, a .nets file containing a list of nets (hyperedges) and the vertices they connect to, a .blk file specifying partitions and their balance constraints, and a .aux file containing one line listing all of the other filenames. Balance constraints are specified by a *target* for each partition, and one *capacity tolerance* as a percent. The balance constraint of a particular partition will be the pair $(\textit{target} * (1 - \textit{capacity\_tolerance}/100), \textit{target} * (1 + \textit{capacity\_target}/100))$. Other optional files may be included, namely a .wts listing vertices and nets with one or more weights, a .fix file which lists vertices and which partitions they are constrained to, and a .sol file which contains an input partitionment. All bookshelf benchmarks may contain commented lines beginning with "#". Solutions are specified by listing all vertices and their partition assignments.

61.7.3 Integrated Circuit Benchmarks from IBM

A set of standard benchmarks derived from VLSI circuits at IBM were presented in ISPD in 1998 [24]. These instances range from 12506 vertices in IBM01 to 210341 vertices in IBM18.

Additionally, partitioning in the context of VLSI placement results in hypergraphs with fixed vertices. Benchmark hypergraphs with fixed vertices were released at ISPD in 1999 [58]. Techniques for evaluating partitioning heuristics and experiments using these benchmarks were given in Ref. [45]. Examples of such comparisons between hMETIS and MLPart are found in Ref. [46].

Appendix: Empirical Results

Figure 61.12 gives runtimes and average solution qualities for 10 runs of MLPart and hMETIS on the ISPD '98 IBM benchmark suite with partitioning tolerances of 2 and 10%. Smaller net cut is better. Runs were performed by a 2.0 GHz Pentium IV Xeon workstation with 2 GB of RAM running Linux.

The size of benchmarks range from 12506 vertices in IBM01 to 210341 vertices in IBM18. MLPart is generally faster than hMETIS, but hMETIS often produces better partitioning results. Both tools produce better solutions when the tolerance is higher. By curve-fitting this data, we find empirically that the runtime and memory usage of both partitioners grow nearly linearly with the size of the benchmark.

We estimate that one can partition a 3.5 million vertex hypergraph in 20 min on a 32-bit machine with the above processor and 4 GB of memory.

Benchmark	Tolerance	Solver	Cut	Runtime	Solver	Cut	Runtime
IBM01	2%	MLPart	240.6	1.725	hMETIS	243	3.402
IBM01	10%	MLPart	231.9	1.535	hMETIS	238.3	3.304
IBM02	2%	MLPart	319.4	3.005	hMETIS	300.3	5.775
IBM02	10%	MLPart	294.4	2.715	hMETIS	296.1	5.159
IBM17	2%	MLPart	2380.2	63.992	hMETIS	2445.5	217.874
IBM17	10%	MLPart	2291.4	62.204	hMETIS	2274.1	221.417
IBM18	2%	MLPart	1840.4	47.253	hMETIS	1687.5	225.259
IBM18	10%	MLPart	1583.5	51.323	hMETIS	1548.7	184.307

FIGURE 61.12 Performance of available software tools on common circuit hypergraph partitioning benchmarks. Results are the average of 10 runs with default configuration. Runs were performed by a 2.0 GHz Pentium IV Xeon workstation with 2 GB of RAM running Linux. Performance of the remaining benchmarks can be found online [57].

References

[1] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.

[2] Kernighan, B. W. and Lin, S., An efficient heuristic procedure for partitioning graphs, *Bell Syst. Tech. J.*, 49, 291, 1970.

[3] Fiduccia, C. M. and Mattheyses, R. M., A linear-time heuristic for improving network partitions, *Proc. DAC*, 1982, p. 175.

[4] Bui, T., Chaudhuri, S., Leighton, F. T., and Sipser, M., Graph bisection algorithms with good average behavior, *Combinatorica*, 7(2), 172, 1987.

[5] Krishnamurthy, B., An improved min-cut algorithm for partitioning VLSI networks, *IEEE Trans. Comp.*, C-33, 438, 1984.

[6] Sanchis, L., Multiple-way network partitioning with different cost functions, *IEEE Trans. Comp.*, 42(12), 1500, 1993.

[7] Hagen, L. W., Huang, D. J., and Kahng, A. B., On implementation choices for iterative improvement partitioning methods, *Proc. Eur. Design Automation Conf.*, 1995, p. 144.

[8] Liu, L. T., Kuo, M. T., Huang, S. C., and Cheng, C. K., A gradient method on the initial partition of Fiduccia–Mattheyses algorithm, *Proc. ICCAD*, 1995, p. 229.

[9] Dutt, S. and Deng W., VLSI circuit partitioning by cluster-removal using iterative improvement techniques, *Proc. ICCAD*, 1996, p. 194.

[10] Alpert, C. J., Huang, J.-H., and Kahng, A. B., Multilevel circuit partitioning, *Proc. DAC*, 1997, p. 530.

[11] Cong, J., Li, H. P., Lim, S. K., Shibuya, T., and Xu, D., Large scale circuit partitioning with loose stable net removal and signal flow based clustering, *Proc. ICCAD*, 1997, p. 441.

[12] Hauck, S. and Borriello, G., An evaluation of bipartitioning techniques, *IEEE Trans. CAD*, 16(8), 849, 1997.

[13] Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S., Multilevel hypergraph partitioning: applications in VLSI design, *Proc. DAC*, 1997, p. 526.

[14] Dutt, S. and Theny, H., Partitioning using second-order information and stochastic gain function, *Proc. ISPD*, 1998, p. 112.

[15] Im, E. J. and Yelick, K. A., Optimizing sparse matrix vector multiplication on SMP, *Proc. PPSC*, 1999.

[16] Ogielski, A. T. and Aiello, W., Sparse matrix computations on parallel processor arrays, *SIAM J. Scientific Comput.*, 14(3), 519, 1993.

[17] Goldberg, M. K. and Burstein, M., Heuristic improvement technique for bisection of VLSI networks, *IEEE Trans. CAD*, 122, 1983.

[18] Ghose, M., Zubairm M., and Grosch, C. E., Parallel partitioning of sparse matrices, *Comp. Syst. Sci. Eng.*, 1, 33, 1995.

[19] Hendrickson, B. and Leland, R., A multilevel algorithm for partitioning graphs, *Proc. Supercomputing*, 1995, pp. 28–41.

[20] Karypis, G. and Kumar, V., Analysis of multilevel graph partitioning, *Technical Report TR 95-037*, Department of Computer Science, University of Minnesota, Minnesota, 1995, pp. 1–19.

[21] Alpert, C. J., Hagen, L. W., and Kahng, A. B., A hybrid multilevel/genetic approach for circuit partitioning, *Proc. ASPDAC*, 1996, p. 298.

[22] Alpert, C. J. and Kahng, A. B., Recent directions in netlist partitioning: a survey, *Integration*, 19, 1, 1995.

[23] Alpert, C. J., The ISPD-98 circuit benchmark suite, *Proc. ISPD*, 1998, p. 80. See errata at `http://vlsicad.ucsd.edu/UCLAWeb/cheese/errata.html`.

[24] Alpert, C. J., Partitioning benchmarks for the VLSI CAD community, `http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html`.

[25] Nam, G., Aloul, F. A., Sakallah, K. A., and Rutenbar, R., A comparative study of two Boolean formulations of FPGA detailed routing constraints, *Proc. ISPD*, 2001, p. 222.

[26] Velev, M. and Bryant, R., Superscalar processor verification using reductions of the logic of equality with uninterpreted functions to propositional logic, in *Correct Hardware Design and Verification methods*, Lecture Notes in Computer Science, Pierre, L., and Kropf, T., eds., Vol. 1703, Springer, Berlin, 1999, pp. 37–53.

[27] Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., and Malik, S., Chaff: engineering an efficient SAT solver, *Proc. Design Autom. Conf. (DAC)*, 2001, pp. 530–535.

[28] Silva, J. and Sakallah, K. A., GRASP: a search algorithm for propositional satisfiability, *IEEE Trans. Comp.*, 48(5), 506, 1999.

[29] Bryant, R., Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Comp.*, 35(8), 677, 1986.

[30] Prasad, M., Chong, P., and Keutzer, K., Why is ATPG easy? *Proc. DAC*, 1999, p. 22.

[31] Aloul, F. A., Markov, I. L., and Sakallah, K. A., Faster SAT and smaller BDDs via common function structure, *Proc. ICCAD*, 2001, p. 443.

[32] Berman, C., Circuit width, register allocation, and ordered binary decision diagrams, *IEEE Trans. CAD*, 10(8), 1059, 1991.

[33] Aloul, F. A., Markov, I. L., and Sakallah, K. A., MINCE: a static global variable-ordering for SAT search and BDD manipulation, *J. Universal Comp. Sci.*, 10(12), 1559, 2004.

[34] Amir, E. and McIlraith, S., Partition-based logical reasoning for first-order and propositional theories, *Artif. Intelligence*, 162(1–2), 49, 2005.

[35] MacCartney, B., McIlraith, S., Amir, E., and Uribe, T. E., Practical partition-based theorem proving for large knowledge bases, *Proc. IJCAI*, 2003.

[36] Caldwell, A. E., Kahng, A. B., and Markov, I. L., Optimal partitioners and end-case placers for standard-cell layout, *IEEE Trans. CAD*, 19(11), 1304, 2000.

[37] Choi, C. and Ye, Y., Application of semidefinite programming to circuit partitioning. Working paper, Department of Management Science, University of Iowa, Iowa, August 1999. `http://citeseer.ist.psu.edu/choi99application.html`.

[38] Donath, W. E. and Hoffman, A. J., Algorithms for partitioning graphs and computer logic based on eigenvectors of connection matrices, *IBM Tech. Disclosure Bull.*, 15(3), 938, 1972.

[39] Eles, P., Peng, Z., Kuchcinski, K., and Doboli, A., System level hardware/software partitioning based on simulated annealing and tabu search, *Design Automation for Embedded Syst.*, 2, 5–32, 1997.

[40] Hur, S. W. and Lillis, J., Mongrel: hybrid techniques for standard cell placement, *Proc. ICCAD*, 2000, p. 165.

[41] Yang, H. H. and Wong, D. F., Efficient network-flow based min-cut balanced partitioning, *IEEE Trans CAD*, 1533, 50–55, 1996.

[42] Alpert, C. J. and Kahng, A. B., Multi-way partitioning via spacefilling curves and dynamic programming, *Proc. DAC*, 1994, p. 652.

[43] Hur, S.-W. and Lillis, J., Relaxation and clustering in a local search framework: application to linear placement, *VLSI Design*, 143, 360–366, 2002.

[44] Caldwell, A. E., Kahng, A. B., and Markov, I. L., Design and implementation of the Fiduccia–Mattheyses heuristic for VLSI netlist partitioning, in *Proc. ALENEX-99, Algorithm Engineering and Experimentation*, Lecture Notes in Computer Science, Goodrich, M. T. and McGeoch, C. C., eds., Vol. 1619, Springer, Berlin, 1999, pp. 177–193.

[45] Caldwell, A. E., Kahng, A. B., Kennings, A. A., and Markov, I. L., Hypergraph partitioning for VLSICAD: methodology for heuristic development, experimentation and reporting, *Proc. DAC*, 1999, p. 349.

[46] Caldwell, A. E., Kahng, A. B., and Markov, I. L., Improved algorithms for hypergraph bipartitioning, *Proc. ASPDAC*, 2000, p. 661.

[47] Karypis, G. and Kumar, V., Multilevel k-way hypergraph partitioning, *Proc. DAC*, 1999, p. 343.

[48] Karypis, G. and Kumar, V., hMETIS: a hypergraph partitioning package version 1.5, user manual, 1998, `http://glaros.dtc.umn.edu/gkhome/fetch/sw/hmetis/manual.pdf`.

[49] Han, E.-H., Karypis, G., Kumar, V., and Mobasher, B., Hypergraph based clustering in high-dimensional data sets: a summary of results, *IEEE Bull. Tech. Comm. Data Eng.*, 21(1), 15–22, 1998.

[50] Cong, J. and Lim, S. K., Edge separability based circuit clustering with application to circuit partitioning, *Proc. ASPDAC*, 2000, p. 429.

[51] Karypis, G., Han, E. H., and Kumar, V., CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling, University of Minnesota, TR #99-007, 1999.

[52] Hagen, L. and Kahng, A. B., A new approach to effective circuit clustering, *Proc. ICCAD*, 1992, p. 422.

[53] MLPart: `http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Partitioning/MLPart/`

[54] hMETIS: `http://www-users.cs.umn.edu/~karypis/metis/hmetis/`

[55] LEF/DEF Language Reference, product version 5.6, Cadence Design Systems, 2004.

[56] UM Physical Design Tools: `http://vlsicad.eecs.umich.edu/BK/PDtools/`

[57] Supplemental illustrations: `http://vlsicad.eecs.umich.edu/BK/PART/illustrations/`

[58] Alpert, C. J., Caldwell, A. E., Kahng, A. B., and Markov, I. L., Hypergraph partitioning with fixed vertices, *IEEE Trans. CAD*, 19(2), 267, 2000.

[59] Karypis, G. and Kumar, V., A parallel algorithm for multilevel graph partitioning and sparse matrix ordering, *Journal of Parallel and Distributed Computing*, 48, 71–85, 1998.

62 Finding Most Vital Edges in a Graph

Hong Shen
University of Adelaide

62.1 Introduction

In many applications such as design of transportation networks, we often need to identify a set of regions/sections whose damage will cause the greatest increase in transportation cost within the network so that we can set extra protection to prevent them from being damaged. Modeling a transportation network with a weighted graph, a set of regions with a set of edges in the graph, transportation cost within the network with a particular property of the graph, we can convert this real-application problem to the following graph-theoretic problem: finding a set of edges in the graph, namely *most vital edges* or *MVE* for short, whose removal will cause the greatest damage to a particular property of the graph. The problems are traditionally referred to as prior analysis problems in sensitivity analysis (see Chapter 30).

Problems of finding the MVE of G for various objectives were addressed in the literature: Lubore et al. [1], Wollmer [2], and Ratliff et al. [3] raised this problem with respect to maximum flow; Corley and Sha [4], Malik et al. [5,6], and Ball et al. [7] considered this problem with respect to shortest paths; Hsu et al. [8] and Lin and Chern [9] addressed this problem with respect to minimum spanning tree (MST). Two central properties of a graph that represent graph's ability of connecting its vertices at minimum cost are MST and shortest paths (SPs). They are particularly important to various network applications. Other graph properties such as diameter and Steiner problem can be derived from them. Recently, finding MVEs with respect to MST and SP has found applications in link failure recovery and replacement SP computation, respectively [10–12.]

In a graph with n vertices and m edges, the problem of finding the k-MVE with respect to MST (the k-MVE problem for short) has important applications in network design and reliability issues, and hence received considerable attention. When $k = 1$, this problem becomes that of finding the single MVE for which a number of results have been developed [8,13–19]. It was shown that the 1-MVE problem can be solved in almost the same time complexity sequentially and same work in parallel on PRAM, as required for computing MST [8,14,17]. For $k \geq 1$, Lin and Chern [9] first showed that a generalized version of the k-MVE problem is NP-hard, Shen [20] first proposed a nontrivial deterministic exact solution using the edge replacement technique, and randomized exact and approximate (2-optimal) solutions. Here we use *exact solution* to mean optimum in quality rather than in efficiency, in order to distinguish from the conventional meaning of *optimal algorithm*. We say that an algorithm is t-optimal if its solution deviates from the exact solution by a factor of at most t. Frederickson and Solis-Oba [21] further proved the NP-hardness of the k-MVE problem and developed a deterministic $O(\log n)$-optimal algorithm. For fixed $k > 1$, in addition to Shen's exact k-MVE algorithm based on edge replacement [20], Liang and Shen [16] presented an algorithm employing the $(k + 1)$-edge certificate of G, Shen [22] showed an improved algorithm that combines edge replacement and edge certificate approaches.

For MVE with respect to single s-t SP, Corley and Sha [4] examined the general k-MVEs problem and gave an algorithm for the case when $k = 1$, referred to here as the *single most vital edge* problem. Malik et al. [5] gave a complexity analysis of the algorithm proposed in Ref. [4], and proposed a more efficient algorithm with the same complexity as that of Dijkstra's SP algorithm for the single MVE problem. The best polynomial bound for Dijkstra's SP algorithm, due to Fredman and Tarjan [23], is $O(m + n \log n)$, where $m = | E |$ and $n = | V |$. Malik et al. [6] also proposed an exponential time algorithm for the k-MVEs problem. This algorithm was shown to be incorrect in their erratum. Bar-Noy et al. [24] showed that a slightly weaker claim corrected from that made in Ref. [5] suffices to ensure the correctness of the algorithm. Sven et al. [25] extended Malik's technique, developed a more efficient algorithm using an auxiliary graph *transmuter* and its parallelization. They have also applied this technique to compute the single MVE with respect to all-pairs SPs (APSP) [26]. Nardelli et al. [27] defined the *detour-critical edge* of a SP as a variant of single MVE and showed that it can be found in the same time complexity as MVE. In their consequent work [28], they applied transmuter in the same way as Ref. [25] to achieve the same complexity of Sven et al.'s algorithm.

In this chapter, we present an overview of deterministic and randomized solutions for the MVE problem with respect to MST and SP. The chapter is organized as follows. Section 62.2 presents exact and approximation solutions for MVE with respect to MST. Section 62.3 presents results for MVE with respect to SP. Section 62.4 concludes the chapter with some remarks.

62.2 MVE with Respect to MST

Given a connected, undirected, and weighted graph $G = (V, E)$ with n vertices and m edges, the k-MVEs of G with respect to MST are a set of k edges in G whose removal results in greatest weight increase in the MST of the remaining graph. The problem is known to be NP-hard for arbitrary k. In this section, we first present a deterministic solution that incorporates the technique of computing replacement edges and constructing an edge certificate of G. We then present randomized solutions to achieve polynomial time complexity.

Let $w(e)$ be a real-valued weight assigned to edge $e \in E$. Denote by $MST(G)$, $E(MST(G))$, and $w(MST(G))$ the MST of G, the set of edges in $MST(G)$, and the total weight of $MST(G)$, respectively. If $K \subset E$ is the set of the k-MVEs of G, then $w(MST(G')) \geq w(MST(G''))$ for any $K' \neq K$ in E, where $G' = (V, E - K)$ and $G'' = (V, E - K')$.

62.2.1 Deterministic Algorithms

We now present the edge replacement and edge certificate approaches used to design deterministic algorithms. We show that combining these two approaches yields a more effective algorithm [22] than that based on either of them. It results in a time-complexity improvement of $O(k)$ factor over the edge certificate approach, and $O(m/(kn))$ factor over the edge replacement approach, respectively, under the usual uniform-distribution assumption that different sizes of the replacement edge-set have an equal probability to occur. We also present a parallel execution of this algorithm on the EREW (Exclusive-Read and Exclusive-Write) PRAM and show that it results in an $\Omega(m/n)$ work reduction over the algorithm using the edge replacement approach. When k is any fixed constant, our algorithm runs in $O(m \log \beta(m, n) + n^{k+1})$ sequential time, and in $O(\log n \log\log n)$ time on an EREW PRAM with $O(n^{k+1})$ processors.

62.2.1.1 Sequential Algorithm

Assume that G is at least $(k+1)$-edge connected. Otherwise any set of k edges containing an edge cut of G would be the k-MVE. We further assume that all the edge-weights are distinct so that, for any $E' \subseteq E$, $MST(G(V, E - E'))$ is unique. Denote by t_S the time required for computing S.

A naive approach requires to consider removal of every possible set K of k edges from m edges of E and computing the MST in the remaining graph, and hence needs time

$$t^{(0)} = \binom{m}{k} t_{MST(G')} \tag{62.1}$$

where $G' = (V, E - K)$.

An edge replacement approach to the k-MVE problem was proposed in Ref. [20]. Let $R(e)$ be the replacement edge set of $e \in MST(G)$ that includes all edges (u, v) in $G - MST(G)$ for which path $P(u, v)$ in $MST(G)$ contains edge e. We compute $R(e)$ for all $e \in E(MST(G))$ by assigning e' to every edge e on the path $P(u, v)$ (i.e., $R(e) \leftarrow R(e) \cup \{e'\}$) for every edge $e' = (u, v) \in E - E(MST(G))$.

Definition 62.1

The replacement edge of $e \in E(MST(G))$, denoted by $r(e)$, is the one with the minimum weight among all edges in $R(e)$. The k replacement edges of $e \in MST(G)$, denoted by $R_e = \{r^1(e), r^2(e), \ldots, r^k(e)\}$, are the k minimum-weighted edges in $R(e)$.

Let $R^+ = \{R_e \mid (\forall e)[e \in MST(G)]\}$ and $R = \cup_{e \in MST(G)} R_e$. Clearly R contains all distinct elements of R^+.

Let Δ_e be R_e's corresponding weight increment array: $\Delta_e[i] = w(R_e[i]) - w(e)$, $1 \le i \le k$. R_e can be obtained by selecting the k smallest-weight elements in $R(e)$ and then sorting them. When R_e is obtained for all $e \in E(MST(G))$, R is a simple matter of computing the union of all R_e. The following procedure computes $R_e, \Delta_e \mid e \in E(MST(G))$.

```
Procedure $k$-ReplacementEdges($G, MST(G), k, \{R_e, \Delta_e \mid e \in E(MST(G))\}$)
    {*Input $G$, $MST(G)$ and $k$; output $R_e, \Delta_e \mid e \in E(MST(G))$ containing all $k$ replacement edges
    and their weight increments of every edge $e \in E(MST(G))$.*}
1.  for every edge $e' = (u, v) \in E - E(MST(G))$ do
        for every edge $e \in E(MST(G))$ on path $P(u, v)$ do
            $R(e) \leftarrow R(e) \bigcup \{e'\}$;
2.  for every edge $e \in E(MST(G))$ do
        Select the $k$ smallest-weight elements in $R(e)$ and store them in $R_e$;
        Sort elements in $R_e$ in increasing order;
3.  for every $e \in E(MST(G))$ do
        for $i = 1$ to $k$ do
            $\Delta_e[i] \leftarrow w(R_e[i]) - w(e)$.
```

This procedure requires $O(mn)$ time: $O(mn)$ for Step 1, $O(n(m + k \log k))$ for Step 2, and $O(kn)$ for Step 3. Since G is k-edge connected, $m \ge k(k-1)/2$ and $k \le n - 1$.

The following lemma proved in Ref. [20] suggests an algorithm to compute the k-MVE.

Lemma 62.1

If K is a set of k-MVE, then the following properties hold:

(1) $K \cap MST(G) \neq \emptyset$;
(2) $K \subset E(MST(G)) \bigcup R$.

Property (2) states that we can exclude all edges in $E - (E(MST(G)) \bigcup R)$ from consideration for the k-MVE. Since $|R| \le k(n-1) \le m$, following Lemma 62.1 we can derive an algorithm that is more efficient than the naive approach.

Denote by t_R and $t_{MST(G)}$ the time required for computing $R = \bigcup_e (R(e))$ and $MST(G)$, respectively. The time required for computing the k-MVE under the edge-replacement approach is then

$$t^{(1)}_{k\text{MVE}} = t_{MST(G)} + t_R + (n-1)\binom{|R|+n-1}{k-1} t_{MST(G')} \tag{62.2}$$

where t_R is the time required for computing R, $n-1$ the number of ways of taking one edge from $E(MST(G))$ by property (1) of Lemma 62.1, $\binom{|R|+n-1}{k-1}$ the number of ways of taking $k-1$ edges from $E(MST(G)) \cup R$ by property (2) of Lemma 62.1, and $G' = (V, E-K)$ for every chosen K.

An alternative approach using the $(k+1)$-edge certificate of G [29] was given in Ref. [16]. As defined in Ref. [30], we define the ith minimum spanning tree of G, T_i, to be

$$T_1 = MST(G); \quad T_i = MST(G - \bigcup_{j=1}^{i-1} T_j), \quad 2 \leq i \leq k+1 \tag{62.3}$$

$U_{k+1} = \cup_{j=1}^{k+1} T_i$ is called the $(k+1)$-edge certificate of G. Clearly U_{k+1} is $(k+1)$-edge connected, since G is $(k+1)$-edge connected. It is known that U_{k+1} can be computed in $O(m)$ [29].

Observe that since U_{k+1} consists of the k edge-disjoint MSTs of G, it must contain both the k-MVE of G and $MST(G(V, E-K))$ for any k edges $K \subset E$. Shen [17] confirmed this for $k=1$ by transforming the maximum spanning tree defined by Iwano and Katoh [14] to the second MST of G. Liang and Shen [16] showed this is also true for arbitrary k and presented an algorithm for computing the k-MVE using the edge certificate approach with time complexity

$$t^{(2)}_{k\text{MVE}} = t_{U_{k+1}} + \binom{(k+1)(n-1)}{k} t_{MST(G'')} \tag{62.4}$$

where $G'' = (V, E(U_{k+1}) - K)$.

Clearly to compare $t^{(1)}_{k\text{MVE}}$ and $t^{(2)}_{k\text{MVE}}$, we need only to compare their dominating parts:

$$(n-1)\binom{|R|+n-1}{k-1} t_{MST(G')} \quad \text{and} \quad \binom{(k+1)(n-1)}{k} t_{MST(G'')}$$

Assume that all sizes of R are equally likely to appear, that is $|R| = s$ with equal probability $1/((n-2)(k-1)+1)$ for all $k-1 \leq s \leq (n-1)(k-1)$. We define this case to be the average case. We have

$$\begin{aligned}(n-1)\binom{|R|+n-1}{k-1} &\leq \binom{|R|+n-1}{k} = \frac{1}{(n-2)(k-1)+1} \sum_{s=k-1}^{(n-1)(k-1)} \binom{s+n-1}{k} \\ &= \frac{1}{(n-2)(k-1)+1}\left(\binom{(n-1)k+1}{k+1} - \binom{n-k-1}{k+1}\right) \\ &= O\left(\binom{(n-1)k}{k} \Big/ k\right)\end{aligned} \tag{62.5}$$

Using Cole et al.'s [31] MST algorithm, we know that

$$\begin{aligned} t_{MST(G')} &= O((m-k)\log\beta(m-k, n)) = O(m\log\beta(m, n)) \\ t_{MST(G'')} &= t_{MST(U_{k+1})} = O(kn\log\beta(kn, n)) \end{aligned}$$

Neglecting the insignificant factor $\log\beta(m, n)/\log\beta(kn, n)$ which is asymptotically constant, we have by Eqs. (62.2) and (62.4)

$$t^{(1)}_{kMVE} \big/ t^{(2)}_{kMVE} = O(m/(k^2 n)) \tag{62.6}$$

Hence the following lemma holds.

Lemma 62.2

For the k-MVE problem, the edge replacement approach is more efficient than the edge certificate approach when $k = \Omega(\sqrt{n})$ *in the average case.*

From the above analysis it is clear that the edge replacement approach reduces the search space for k-MVE better than the edge certificate approach ($|R| \leq |U_{k+1}|$), whereas the edge certificate approach reduces the remaining graph for computing the weight increment (MST) after removal of each set of k edges better than the edge replacement approach ($|U_{k+1}| \leq |G'|$). So a better algorithm is to incorporate both approaches [22]. The resulting algorithm computes both R and U_{k+1}. It uses the edge replacement approach to reduce the search space for the k-MVE from $\binom{m}{k}$ to $(n-1)\binom{|R|+n-1}{k-1}$, and uses the edge certificate approach to reduce G' to U_{k+1}. The algorithm is given below.

```
Algorithm Exact-k-MVE
    {*Input $G$ and $k$; output $k$-MVE containing the $k$-MVEs of $G$.*}
1.  Compute $T_1 = MST(G)$;  $\Delta \leftarrow 0$
    for $i = 1$ to $k$ do
        $E' \leftarrow E' \cup E(T_i)$; Compute $T_{i+1} = MST(G(V, E - E'))$;
    {*Compute $U_{k+1}$; $\Delta$ is the total weight increment introduced by removal of $k$ edges from $G$.*}
2.  $k$-ReplacementEdges($\cup_{i=2}^{k} T_i$, $MST(G)$, $k$, $\{R_e, \Delta_e \mid e \in E(MST(G))\}$);
    {*Compute $R$.*}
3.  for every edge $e_1 \in E(MST(G))$ do
        for every edge-set $\{e_2, \ldots, e_k\} \in (E(MST(G)) - \{e_1\}) \cup R$ do
            Remove $\{e_1, \ldots, e_k\}$ from $E(MST(G)) \cup R_e$;
            Compute $MST(G')$ for $G' = (V, E(U_{k+1}) - K)$;
            if $w(MST(G')) - w(MST(G)) > \Delta$ then
                Mark $\{e_1, \ldots, e_k\}$ as the current $k$-MVE.
```

Time complexity of the algorithm: Step 1 requires time $O(km \log \beta(m, n))$ by Ref. [31]. Step 2 requires $O(kn^2)$ time, since $| \cup_{i=2}^{k} T_i| = (k-1)(n-1)$. There are $\binom{|R|}{k}$ iterations in Step 3, each taking time $t_{MST(U_{k+1})}$, so the total time required for Step 3 is $O((n-1)\binom{|R|+n-1}{k-1} kn \log \beta(kn, n))$.

By Eq. (62.5) $\binom{|R|}{k} = \binom{k(n-1)}{k}/k$ in the average case. The size of R is $k-1$ in the best case and $(k-1)(n-1)$ in the worst case. So we have the following theorem.

Theorem 62.1 [22]

Given a connected, undirected and weighted graph G with n vertices and m edges, we can find the k-MVEs in G in

- $O(km \log \beta(m, n) + (n-1)\binom{(k+n-2)}{k-1} kn \log \beta(kn, n))$ *in the best case;*
- $O(km \log \beta(m, n) + (n-1)\binom{k(n-1)}{k-1} n \log \beta(kn, n))$ *in the average case;*
- $O(km \log \beta(m, n) + (n-1)\binom{k(n-1)}{k-1} kn \log \beta(kn, n))$ *in the worst case.*

The above theorem shows that with the same worst case time complexity, our algorithm has an $O(k)$ factor speedup to the algorithm using the edge certificate approach in the average case. By Eq. (62.6), it is clear that our algorithm is $O(m/(kn))$ factor faster than the algorithm using the edge replacement approach in the average case.

Since $\binom{m}{k} < (m-k+1)^k$ holds for any $k \leq m$, the following corollary is immediate.

Corollary 62.1

The k-MVEs in a connected, undirected, and weighted graph G with n vertices and m edges can be computed in $O(km \log \beta(m, n) + (n-1)(kn+2)^{k-1} kn \log \beta(kn, n))$ time.

When k is a fixed constant, the above complexity can be simplified:

Corollary 62.2

For any fixed $k \geq 1$, the k-MVEs in a connected, undirected, and weighted graph G with n vertices and m edges can be computed in $O(m \log \beta(m, n) + n^{k+1})$ time.

62.2.1.2 Parallelization

Given algorithm `Exact-`k`-MVE`, we now show how to implement it on the EREW PRAM.

Applying Chrong et al.'s MST algorithm [32] $k+1$ times, each producing a T_i, we can implement Step 1 in $O(k \log n \log\log n)$ time with $O(m)$ processors. At the ith time we need to mark all the edges in $\cup_{j=0}^{i} T_i$, which can be done obviously in $O(1)$ time with m processors.

Having obtained $U_{k+1} = \cup_{j=1}^{k+1} T_i$, we first make $k(n-1)$ copies of $T_1 = MST(G)$ for the purpose of exclusive read, and allocate one processor to each edge in U_{k+1} and use an array L_e of size $k(n-1)$ at each edge in $MST(G)$, where $L_e[i]$ is used for keeping the label (if there is) to be written by processor i, $1 \le i \le k(n-1)$. Then we use the path labelling procedure based on the techniques of Euler tour (for finding the lowest common ancestor of vertex pair (u, v) which divides path $P(u, v)$ into two upward subpaths) and pointer jumping (for propagating the label along the subpaths) [17], where at each step of writing different processors write their labels into different cells in array L_e associated at edge $e \in E(T_1)$. This requires $O(\log n)$ time and $k(n-1)$ processors. Now we allocate $k(n-1)$ processors to each L_e and sort its nonempty cells in increasing order. Clearly $R = \bigcup_{i=1}^{n-1} L[1 \ldots k-1]$. Hence Step 2 can be completed in $O(\log n)$ time using $O(kn^2)$ processors.

Now we make $(n-1)\binom{|R|+n-1}{k-1} = O((n-k+1)^k)$ copies of U_{k+1} and allocate $k(n-1)$ processors to each copy. For every copy in parallel we compute the k edges to be removed (K), that is, we compute all the combinations of $(n-1)\binom{|R|+n-1}{k-1}$ using this number of processors, which can be done in $O(k)$ time by using Knott's numbering system for combinations [33]. We then in parallel compute the MST in every copy of U_{k+1} *using the edge certificate approach* after removal of its K in $O(\log n \log \log n)$ time using $k(n-1)$ processors per copy, and find the minimum weight increment among them in $O(k \log n)$ time using $O(n^k)$ processors. So Step 3 requires $O((k + \log \log n) \log n)$ time using $O(kn(n-k+1)^k)$ processors.

Hence we have the following theorem.

Theorem 62.2 [22]

Given a connected, undirected, and weighted graph G with n vertices and m edges, using $O(kn(n-k+1)^k)$ processors on an EREW PRAM, we can find the k-MVEs in G in $O((k + \log \log n) \log n)$ time.

When k is a fixed constant, the above complexity has a simple form:

Corollary 62.3

For any fixed $k \ge 1$, the k-MVEs in a connected, undirected, and weighted graph G with n vertices and m edges can be computed in $O(\log n \log \log n)$ time on an EREW PRAM with $O(n^{k+1})$ processors.

Clearly, our algorithm is an *NC* algorithm for any fixed k. In comparison with Liang and Shen's algorithm [16], our algorithm has a $\Theta(m/n)$ factor work-reduction. Note that in Ref. [16] the claimed time complexity $O(\log n)$ excludes the time required for computing $MST(G)$ which is $O(\log n \log \log n)$ on EREW PRAM.

62.2.2 Randomized Solutions

Since the k-MVE problem is NP-hard for arbitrary k, our interest should turn to seeking polynomial-time randomized solutions. In this section, we outline two randomized algorithms of Ref. [20] that produce optimal and near-optimal solutions with high probability in polynomial time.

62.2.2.1 Randomized Optimal Solution

For any $x \in R$, we define the *rank* of x in R_e, denoted by $rank_e(x)$, to be i if x is the ith element in R_e. We call $rank(x) = \min_e\{rank_e(x)\}$ the rank of x. Let $R^i = \{x \in R \mid (\exists e)[rank_e(x) = i]\}$ and Δ^i be R^i's corresponding weight increment array for $i = 1, \ldots, k$. Clearly R^i contains a set of replacement edges, each being the result of removal of i edges ($i-1$ edges before it in its R_e and $\{e\}$). When the total weight increment introduced to the MST is maximum possible, R^i becomes the result of removal of a set of k-MVE. The k-MVE can be computed by identifying a set of replacement edges in $\cup_i R^i$ that introduces a maximum weight increment to the MST. We call such a set of edges *maximum replacement edges.*

Assume that $R_e \cap R_{e'} = \emptyset$ for any $e \neq e' \in E(MST_G)$. We say that R^j *improves* R^i, denoted by $R^j \triangleright R^i$, if replacing a subset of R^i with a subset of R^j corresponding to an equal number of edges to be removed increases the weight increment. It is clear that if we keep elements in every R^i sorted in decreasing order, then a maximal (weight increment) improvement on R^i by R^j must be resulted by replacing a (contiguous) segment of elements from the last (smallest) toward the first (largest) in R^i with a segment from the first toward the last in R^j. We call such segments in R^i and R^j the *tail* and *head*, denoted by $T(x)$ and $H(x)$ for length x, respectively. By the definition of improvement apparently a segment $T(x)$ of R^i can be *legally* replaced by a segment $H(y)$ of R^j only when $y = ix/j$. We call $H(y)$ the *improver* of R^i, denoted by $H(x) \triangleright R^i$. It is not hard to observe that given R^i, $i = 1, \ldots, k$, a set of maximum replacement edges can be obtained via repeatedly improving any R^i by R^j for all $j \neq i$.

Theorem 62.3 [20]

Given a connected, undirected, and weighted graph G with n vertices and m edges, we can compute the k-MVEs of G with respect to MST in time $O(mn)$ with probability at least $e^{-\frac{k^2}{2(m-n-1)} - \frac{2\log_c k}{k-4}}$, where $c = 1 + \frac{1}{2^{k/2}}$.

62.2.2.2 Randomized Approximate Solution

An algorithm is called t-optimal if its output differs from the optimal solution within a factor t. Study on approximate algorithms is an important contemporary trend on algorithms research, because for many problems optimal (exact) solutions are very hard to obtain whereas t-optimal solutions may be easily obtainable within polynomial time. Here we present a randomized algorithm that produces 2-optimal solution to the k-MVE problem. Our algorithm sets up a "cutoff" point for the weight increment and considers only those replacement edges whose weight increments are not below the cutoff point. The algorithm runs in multiple phases, each with a reduced cutoff point by a factor 2, starting from the maximum possible cutoff point and ending at the minimum possible cutoff point. This first randomization that neglects the possibility that the combination of some replacement edges below the cutoff point and some above the point can also result in the overall weight increment within the specified bound. This probability is, however, quite small $\left(\sum_{i=1}^{k-1} \binom{x}{i} \binom{kn-x}{k-i} / \left(k \binom{kn}{k} \right) \right)$ if there are x replacement edges below the cutoff point).

Let $\Delta^i_{\max} = \max_j \{\Delta^i_j \mid j \in R^i\}$ and $\bar{\Delta} = \sum_{i=1}^{k} \Delta^i_{\max}/k$. Clearly $\bar{\Delta}$ is the maximum possible cutoff point of weight increment per edge for k replacement edges, and $\bar{\Delta}\frac{k}{k-i}$ is the maximum possible cutoff point for $k-i$ replacement edges, $1 \leq i \leq k-1$. Given cutoff point $\bar{\Delta}$, let $\tilde{R} = \{x \in R \mid \Delta_x \geq \bar{\Delta}\}$ and $I = \{\tilde{e}_i \mid \tilde{R}[i] \in R_{\tilde{e}_i}, 1 \leq i \leq |\tilde{R}|\}$. We define the *fringe* of $\tilde{R}$, $F_{\tilde{R}}$, to be a (rank) ordered subset of $\tilde{R}$ that contains, for every distinct element $\tilde{e}$ in I, the corresponding element in $\tilde{R}$ that has the smallest rank in $R_{\tilde{e}}$. Let $I' = \{\tilde{e}'_i \mid F_{\tilde{R}}[i] \in R_{e'_i}, 1 \leq i \leq |F_{\tilde{R}}|\}$. Obviously if $i < j$ then $I'[i] \neq I'[j]$ and $rank_{I'[i]}(F_{\tilde{R}}[i]) < rank_{I'[j]}(F_{\tilde{R}}[j])$. The second randomization to be made here is that we assume that, for any $i < j$, $F_{\tilde{R}}[j]$ can be included into the replacement edge set satisfying the conditions only when $F_{\tilde{R}}[i]$ has already been included. It is obvious that $F_{\tilde{R}}[i]$ has a much greater probability to be chosen than $F_{\tilde{R}}[j]$ as $rank_{I'[i]}(F_{\tilde{R}}[i]) < rank_{I'[j]}(F_{\tilde{R}}[j])$. The reverse can happen only when $R_{I'[j]}[1 \ldots rank_{I'[j]}(F_{\tilde{R}}[j])]$ has an extremely high ratio of duplicates with others, which has a very low probability too. This suggests that the replacement edge satisfying the conditions w.r.t. the current cutoff point, if exists, should be $F_{\tilde{R}}[1 \ldots x]$ for some x.

Theorem 62.4 [20]

For computing the k-MVEs with respect to minimum spanning tree in a connected, undirected, and weighted graph G with n vertices and m edges, we can obtain an optimal solution with probability of success at least $e^{-\frac{\sqrt{2k}}{k-2}}$ in $O(n)$ time using n^2 processors, and an approximate two-optimal solution with probability of success at least $1 - \frac{1}{4}(\frac{2}{n})^{k/2-2}$ in $O(\log^2 n)$ time using $mn/\log n$ processors, on a CREW PRAM.

62.3 MVE with Respect to SPs

Given a connected, undirected, and weighted graph $G = (V, E)$, a set of edges is called MVEs with respect to shortest paths if removing them from G will result in greatest weight increase for the SPs concerned in the remaining graph. In this part, we address the problems of computing the single MVE with respect to the s-t SP and APSPs ORS, respectively. We present the algorithms developed in our previous work for solving these problems and show the schemes for their efficient parallelization [25,26].

We assume without loss of generality that a source node s and a terminal node t are at least two-edge connected. If s and t are not two-edge connected, then any edge whose removal results in the disconnection of s from t is a single MVE of G with respect to SPs.

62.3.1 MVE with Respect to Single Source SPs

For each edge $(i, j) \in E$, assume a length (weight) $d(i, j)$ is associated with it. For any two nodes s and $t \in V$, let P be the SP from s to t in G.

An edge e is the single MVE with respect to P if the length of the shortest s-t path in $G(V, E\backslash\{e\})$ is at least the length of the shortest s-t path in $G(V, E\backslash\{e'\})$, for all $e' \in P$.

In this section, we describe our sequential $O(m\alpha(m, n))$ time algorithm for the single MVE problem [25] for a weighted undirected graph $G = (V, E)$, where $n = |V|$ and $m = |E|$. The time complexity of our algorithm is asymptotically equivalent to Malik et al.'s algorithm [5]. However, unlike Malik's algorithm, our algorithm is not inherently sequential and thus lends itself to efficient parallelization.

We show how to parallelize our algorithm in $O(\log n)$ time using m processors on the CRCW (Concurrent-Read and Concurrent-Write) PRAM computational model and $O(m)$ space, $O(\log n)$ time using $mn/\log n$ CREW (Concurrent-Read and Exclusive-Write) processors and $O(m + n \log m)$ space, and in $O(\log n)$ time and $O(mn)$ space using $mn/\log n$ EREW processors. These are the first NC algorithms for this problem to the best of our knowledge.

62.3.1.1 Sequential Algorithm

It is clear that the most vital edge with respect to the shortest s-t path of a graph G must lie on the shortest s-t path. Let T be a tree of SPs from s to all other nodes. Let P be the shortest s-t path in T, $u(i)$ be the shortest distance from s to i, and let $v(i)$ be the shortest distance from i to t. Let T_s and T_t be the two subtrees of T created by the removal of an edge $e \in P$. Note that T_s contains s, and T_t contains t.

The sequential algorithm given by Malik et al. [5] is based on the following observations.

Observation 62.1

An edge (i, j) is on a shortest s-t path if and only if

$$u(t) = v(s) = u(i) + d(i, j) + v(j) = \mathbf{min}_{(x,y)\in E}\{u(x) + d(x, y) + v(y)\} \tag{62.7}$$

Observation 62.2

Let T be a tree of SPs from s to all the nodes and let P be the shortest s-t path in T. If some edge $e \in P$ is removed from T, dividing the node set V into V_s and $\overline{V_s}$ such that $s \in V_s$ and $t \in \overline{V_s}$, then there exist SPs from all other nodes in $\overline{V_s}$ to t that do not use the edge e.

Observation 62.2 was given by Bar-Noy et al. [24] by correcting an erroneous claim made in Ref. [5].

Let $Q(i, j) = \{(x, y) \in E \backslash (i, j) : x \in T_s \text{ and } y \in T_t\}$. It is clear that any path from s to t in the absence of (i, j) must have an edge in this cut $Q(i, j)$. It is shown in Ref. [5] that using Observations 62.1 and 62.2, we may find this edge and the length of the shortest s-t path in $G(V, E\backslash\{i, j)\})$ by computing the following formula:

$$\mathbf{min}_{(x,y)\in E}\{u(x) + d(x, y) + v(y)\} \mid (x, y) \in Q(i, j) \tag{62.8}$$

We can identify the single MVE if we compute (62.8) for every edge $(i, j) \in P$. Finding the MVE w.r.t. the s-t path will thus require $O(m|P|)$ time. Malik et al. [5] reduced this to $O(m + n \log n)$ using Fibonacci heaps [23]. However, their algorithm cannot be parallelized efficiently due to its inherently sequential nature. For the purpose of efficient parallelization, we presented another sequential algorithm using a different approach based on the following observation [25]:

Observation 62.3

Every nontree edge (x, y) creates a cycle with a tree edge (i, j) if and only if $(x, y) \in Q(i, j)$.

Proof

Consider an edge $(i, j) \in T$, and an edge $(x, y) \notin T$. We say an edge (x, y) *spans* T_s and T_t if its two endpoints reside in T_s and T_t respectively. Let (x, y) span T_s and T_t if (i, j) is removed from T. We can find the *nearest common ancestor* of x and y ($NCA(x, y)$), that is, the node at which the paths from x to s and from y to s in T converge. T is rooted at s, and the component T_s contains s, thus, $NCA(x, y) \in T_s$. Obviously, there is only one edge (i, j) in T that spans T_s and T_t. Hence, any edge (x, y) that spans T_s and T_t must create a cycle with (i, j). Furthermore, any edge (x, y) not spanning T_s and T_t does not create a cycle with (i, j) as both x and y must be in either T_s or T_t, which implies that $NCA(x, y)$ is also in the same component. □

We use an auxiliary graph called a *transmuter* [34] to simultaneously compute the set of nontree edges $Q(i, j)$ for each tree edge (i, j). A transmuter is directed and acyclic, and represents the set of fundamental cycles of G with respect to T. For a transmuter D, we name the vertices of D *nodes* to avoid confusion with the vertices of G. In a transmuter, *source* nodes and *sink* nodes have in-degree zero and out-degree zero, respectively. For each tree edge e, the transmuter has a corresponding source node $s(e)$, and for each nontree edge f, there is a sink node $t(f)$. There are an arbitrary number of additional nodes between the source and sink nodes. There is a path from a source node $s(e)$ in D to a sink node $t(f)$ if and only if the edge e lies on the tree-path $T(f)$. By Observation 62.3, and the definition of transmuter, a transmuter will associate all nontree edges (x, y) with each tree edge (i, j) if (i, j) is on the tree path $T(x, y)$, thus making simultaneous computation of the set of nontree edges $Q(i, j)$ possible.

Our sequential algorithm follows Tarjan [34] SP sensitivity analysis algorithm by labeling each edge in $Q(i, j)$ with the length of s-t SP calculated by Observations 62.1 and 62.2 instead of the cost of a replacement edge in Tarjan's algorithm. A high level description of our algorithm from Ref. [25] is given below.

```
Algorithm Single-SP-MVE
    {*Input: A weighted undirected graph $G = (V, E)$ and a shortest path tree $T$; Output: The MVE
    wrt the s-t SP.*}
1.  Calculate $u(x)$ and $v(x)$ for every vertex $x$ in $T$;
2.  Compute the path length $L(f)$ from $s$ to $t$ where $L(f) = d(x, y) + u(x) + v(y)$
    for every nontree edge $f = (x, y)$;
3.  Compute the nearest common ancestor $NCA(x, y)$ for every nontree edge $(x, y)$;
4.  Calculate the auxiliary edge $f'$ for each nontree edge $f$;
    {*The auxiliary edge $f'$ for a nontree edge $f = (x, y)$ is defined as $(NCA(x, y), y)$.*}
5.  Construct a transmuter for the auxiliary graph $G' = (V, E')$,
    where $E' = \{e \mid e$ is a tree edge $\} \cup \{ f' \mid$f is a nontree edge $\}$;
6.  Label each sink $f'$ of the transmuter with $L(f)$;
7.  Process the nodes of the transmuter in reverse topological order:
        For any node that is not a sink, label it with the minimum of
        the labels of its immediate successors;
8.  Find the maximum label for each edge $(i, j) \in P$ from Step 7.
```

Steps 1 and 2 ensure the correct value for the labelling of the sink nodes in Step 6. Steps 3–5 ensure that structural correctness of the transmuter. Step 7 ensures that we choose the minimum label among all possible labels for a source node. Step 8 identifies the MVE.

In Step 2, if we assume that the APSP information is available, then the function $v(y)$ for every vertex $y \in V$ can be calculated in the same manner and time as $u(y)$. Step 6 can be done in $O(n)$ time by simply taking the maximum label over all tree edges on the s-t SP. Step 8 requires $O(n)$ time. So the time complexity of our algorithm is $O(m\alpha(m, n))$ as all other steps are the same as those in Tarjan's SP sensitivity analysis algorithm.

62.3.1.2 Parallel Algorithm

Based on the sequential algorithm proposed in the previous section, the following parallel algorithm can be easily derived [25].

```
Algorithm Par-Single-SP-MVE
    {*Input: a weighted undirected graph $G = (V, E)$ and a SP tree $T$; Output: the MVE w.r.t. the s-t SP.*}
1. Compute $u(x)$ and $v(x)$ for every vertex $x$ in $T(s)$ in parallel;
2. Compute the path length $L(f)$ from $s$ to $t$ where $L(f) = d(x, y) + u(x) + v(y)$
   for every nontree edge $f = (x, y)$ in parallel and label $(x, y)$ with $L(f)$;
3. Using the path labeling technique from Ref. [15] and the construction in Section 62.3,
   compute the minimum label for each tree edge in parallel:
   3.1 For each nontree edge $(x, y)$, project its label to every tree
       edge on the cycle $T(s) \bigcup \{x, y\}$;
   3.2 For each tree edge find the minimum label among all labels
       assigned to it in Step 3.1;
4. Find the maximum label among all edges $(i, j) \in P$ in parallel.
```

We now show its implementations on different PRAM models.

62.3.1.2.1 CRCW Implementation

The computational model for the algorithm is the MINIMUM-CRCW PRAM. On this model, if a write conflict arises, the processor holding the minimum value is allowed to write. This processor writes its value if and only if the value to be written is smaller than any previously computed value.

Given a MINIMUM-CRCW PRAM with m processors, we can implement Tarjan's Shortest Path Sensitivity Analysis algorithm [34] in $O(\log n)$ time by a path labelling technique described in Ref. [15]. This combines the techniques of Euler tour and list ranking using pointer jumping [35] to find the nearest common ancestor of a nontree edge (x, y) and assign the correct label to each edge on the paths from x and y to that ancestor. The Euler tour technique is used to split the path P from x to y into two subpaths by finding the lowest common ancestor of vertices x and y. We then use list ranking with pointer jumping to propagate the labels along the two subpaths.

Note that the main change we have made to the Shortest Path Sensitivity Analysis algorithm is to calculate the $v(x)$ values for every vertex x, and to enumerate the replacement shortest path lengths. Also, we need to find the maximum label over all edges $(i, j) \in P$. This maximum label and its associated edge will be the MVE with respect to the s-t path.

In Step 1, we assume that we are given the APSP information, and this enables us to compute the $v(x)$ values in the same time and manner as the $u(x)$ values. It was shown that this computation requires $O(\log n)$ time using n processors [15]. Step 2 may be done in constant time on m processors as there are at most $m - n + 1$ nontree edges. Step 3 can be done in $O(\log n)$ time using m processors. Step 4 can be accomplished in $O(\log n)$ time on n processors. So the complexity of the above algorithm on the MINIMUM-CRCW PRAM is $O(\log n)$ using m processors. Clearly the algorithm requires $O(m)$ space.

62.3.1.2.2 CREW Implementation

We now show how to implement algorithm Single-SP-MVE on a CREW PRAM with $mn/\log n$ processors in $O(\log n)$ time.

We implement Step 1 in $O(\log n)$ time using n processors on an EREW PRAM as follows. To compute $u(x)$, $x \in V$, we root T at s using the Euler tour technique as in Ref. [35, Section 3.2], which requires

$O(\log\ n)$ time on n processors on the EREW PRAM. However, instead of assigning a weight of 1 to each arc (x, y) on the Euler tour, we assign $d(x, y)$. This will give us the distance of each vertex from s. We follow similar steps to compute $v(x)$, $x \in V$. For Step 2 we allocate one processor to each nontree edge f and compute the path length $L(f)$. This can be done in constant time using m processors as there is no interprocessor communication and hence, no write conflict.

For Step 3 we use an analogous construction to that used in Shen [17], which applies the path labelling technique on a CREW PRAM. We split Step 3 into two parts. First we compute the labels for each edge $e \in T$, then we find the minimum label for each tree edge e. To compute the labels for each tree edge we proceed as follows. We allocate one processor to each nontree edge f and use an array A_e of size $m - n + 1$ for each edge $e \in T$. We store the label for a processor i at position $A_e[i]$, where $1 < i \leq m - n + 1$. In effect we are creating $m - n + 1$ copies of T where each processor can work on its own copy. Note that this requires $O(mn)$ space. We use the same labelling technique so this step takes $O(\log\ n)$ time using m processors.

To compute the minimum label for each tree edge e we allocate $m/\log\ n$ processors to e. Using standard techniques, we find the minimum label among all labels for a tree edge e as follows. Divide A_e into $m/\log\ n$ subarrays of size $\log\ n$. Find the minimum element in each subarray using one processor per subarray. Next find the global minimum among the $m/\log\ n$ local minima using one processor per local minimum. This can be done in $O(\log\ n)$ time using $nm/\log\ n$ processors.

It is shown in Ref. [17] that the space requirement can be reduced by compressing each A_e into m bits and creating a new array A' of size m to store the labels for each nontree edge. So, Step 3 may be done in $O(\log\ n)$ time using $nm/\log\ n$ processors.

Finding the maximum among $n - 1$ values can be done in $O(\log\ n)$ time using n processors on this model. This gives an $O(\log\ n)$ algorithm using $nm/\log\ n$ CREW processors and $O(m + n\log\ m)$ space.

62.3.1.2.3 EREW Implementation

We use a technique as shown in Ref. [17] to show that our EREW algorithm can also be implemented on the EREW PRAM with the same time and processor complexity using $O(mn)$ space.

Step 1 is already shown to require $O(\log\ n)$ time on m processors in the CREW implementation. To remove concurrent read conflicts we create $m - n + 1$ copies of T, and assign a copy to each nontree edge e. This requires $O(mn)$ space. Note that we must use the uncompressed version of A_e for each tree edge e to resolve concurrent read conflicts. Hence, we can implement our CREW algorithm on the EREW PRAM in $O(\log\ n)$ time using $mn/\log\ n$ processors with $O(mn)$ space.

Several efficient parallel algorithms were also developed for this problem [15,17]. Due the inherent sequentiality of the single MVE algorithm proposed by Malik et al., it was not possible to develop an efficient parallel algorithm based on their approach.

62.3.2 MVE with Respect to APSPs

For the APSPs problem we define two types of *MVEs* based on two different measurements.

MVE1. An edge e^* is most vital if for all other $e \in E$, $\max_{i,j \in V}\{dist_{G^*}(i, j) - dist_G(i, j)\} \geq \max_{i,j \in V}\{dist_{G'}(i, j) - dist_G(i, j)\}$, where $G^* = (V, E \backslash \{e^*\})$ and $G' = (V, E \setminus \{e\})$.

MVE2. An edge e^* is most vital if for all $e \in E$, $\sum_{i,j \in V} dist_{G^*}(i, j) \geq \sum_{i,j \in V} dist_{G'}(i, j)$.

For MVE1, we wish to identify the highest increase in the distance between any two vertices s and t, and not the maximum length of any $s - t$ path. Hence, the presence of the $dist_G(i, j)$ term on each side of the comparison. For MVE2, we compute the highest increase in the total (summed) weight for all pairs shortest paths.

The *diameter* of a graph $G = (V, E)$ is defined as the maximum length of the shortest paths among all pairs of vertices in V (see, e.g., [36,37]). For the diameter problem, we define the MVE as the edge which when removed causes the greatest increase in the length of the diameter. It is not difficult to see that an almost straightforward extension of MVE1 can result in a solution to MVE with respect to the diameter.

In this section, we first present our sequential algorithm for computing MVE w.r.t. to APSPs [26], which is an extension of the algorithm of MVE w.r.t. to single SP introduced in the previous section. We then show how to implement this algorithm in parallel in $O(\log\ n)$ time for both MVE1 and MVE2 using mn^2 CRCW processors.

62.3.2.1 Sequential Algorithm

To solve MVE1 and MVE2 we extend the result in Ref. [25] as follows. We perform a preprocessing step that associates with each edge $e \in T^{\omega}$ the set $V(\overline{T}_e^{\omega})$ of vertices to which a new path from ω must be calculated. We label each tree edge e with the vertex $v \in (V\backslash\{\omega\})$ if $e \in E(P(\omega, v))$ to determine which vertices to reconnect to ω when e is removed. We basically run the algorithm of Ref. [25] once for each vertex pair, for a total of n^2 times. Finally, we calculate the maximum increase in any inter-vertex distance for MVE1, and the change in the weight of the n APSP trees for MVE2. Note that we may use the algorithm for solving MVE1 to compute the MVE with respect to the diameter of the graph. We first compute the length of the replacement path for each vertex pair that is disconnected when a tree edge is removed. Then, instead of computing the change in vertex distances, we find the maximum distance between each pair of vertices. This gives us the maximum distance between any pair of vertices, and hence, the new diameter of the graph.

We first give a procedure Pre-MVE, which we use to preprocess the n APSP trees. This procedure is used for both MVE1 and MVE2.

```
**Procedure** Pre-MVE($T^z | \bar{z} \in V$, $V(\overline{T}_e^{\omega})$)
    {*Input: $n$ SP trees $T^z$ rooted at $z$ for each $z \in V$; Output: the MVE w.r.t. the s-t shortest path;
    Output: the sets $V(\overline{T}_e^{\omega})$ of nodes disconnected when $e$ is removed from $T^{\omega}$*}
1.  for each $\omega \in V$ do
            for each $e \in T^{\omega}$ do
                Calculate $V(\overline{T}_e^{\omega})$
                {*The set of vertices to reconnect with $\omega$ when $e$ is removed from $T^{\omega}$.*}
```

We now give two procedures Post-MVE1 and Post-MVE2, which generate MVE1 and MVE2, respectively, after all replacement paths have been calculated.

```
**Procedure** Post-MVE1
    for each tree edge $e$ do
            $M(e) = \mathrm{MAX}\ \{dist_{G=(V,E\backslash\{e\})}(\omega, j) - dist_G(\omega, j), e \in T^{\omega}$, and $j \in V(\overline{T}_e^{\omega})\}$;
    MVE1 = MAX($M(e)$).

**Procedure** Post-MVE2
    For each tree edge $e$ do
            $S(e) = \sum\ \{dist_{G=(v,E\backslash\{e\})}(\omega, j) - dist_G(\omega, j), e \in T^{\omega}$, and $j \in V(\overline{T}_e^{\omega})\}$;
    MVE2 = MAX($S(e)$).
```

The main idea of the sequential algorithm is to first find the set of nodes $V(\overline{T}_e^{\omega})$ disconnected when each tree-edge e is removed. This is done using procedure Pre-MVE. Then each nontree edge has an $n \times n$ matrix of inter-vertex replacement paths computed. Each element of this $n \times n$ matrix is propagated up the transmuter structure. Recall that the transmuter structure associates all nontree edges (x, y) with each tree-edge e if and only if e is on the cycle created when (x, y) is added into the tree. We keep only the minimum value computed so far for each element of the matrix. Once the propagating stage is complete, we use either procedure Post-MVE1 or procedure Post-MVE2 to solve problem MVE1 or MVE2, respectively.

The extended algorithm from Ref. [25] is:

```
**Algorithm** All-Pairs-MVE
1. Call Procedure Pre-MVE;
2. Calculate $dist(\omega, x)$ and for every vertex pair $\omega$ and $x$ in $V$;
3. Compute an $n \times n$ matrix $A(f)$, for each nontree edge $f = (x, y)$, and assign each element $[i, j]$
   of the matrix with $d(x, y) + dist(i, x) + dist(j, y)$;
4. Compute the nearest common ancestor $nca(x, y)$ in each SP tree for every nontree edge $(x, y)$;
```

```
 5. Calculate the auxiliary edge $f'$ for each nontree edge $f$ in each SP tree;
    {*The auxiliary edge $f'$ for a nontree edge $f = (x, y)$ is defined as $(nca(x, y), y)$. We construct
    the transmuter for each SP tree using the auxiliary edges $f'$ and each tree edge. Note that $f'$ is a
    single edge, but it may represent multiple edges on the cycle created with tree edge $f$.*}
 6. Construct a transmuter for the auxiliary graph $G' = (V, E')$,
    where $E' = \{e \mid e$ is a tree edge $\} \cup \{ f' \mid$f is a nontree edge $\}$;
 7. Label each sink $f'$ of the transmuter with matrix $A(f)$;
 8. Each sink $f'$ will now have $n^2$ labels associated with it, each of which represents a replacement path
    for a vertex pair;
 9. Process the nodes of the transmuter in reverse topological order:
    For any node that is not a sink, label it with the minimum of the labels of its immediate successors.
    That is, node $v_i$ receives the minimum of the $i$th label of each of $v$'s immediate successors;
10. Call either procedure Post-MVE1 or Post-MVE2 to calculate MVE1 or MVE2, respectively.
```

Step 1 finds the sets $V(\overline{T}^{\omega}_{e})$ of vertices that are disconnected from the root $\omega \in V$ of each tree when an edge e is removed from T^{ω}. Steps 2 and 3 ensure the correct value for the labeling of the sink nodes in Step 7. Steps 3–6 ensure the structural correctness of the transmuter. Step 8 ensures that we choose the minimum label among all possible labels for a source node, and Step 9 finds the MVE (the tree edge with the maximum label) with respect to either MVE1 or MVE2.

For brevity, we do not discuss the complexity of the above sequential algorithm. For a detailed discussion see Refs. [25,34].

In what follows we present the implementation of this algorithm on the CRCW PRAM model.

62.3.2.2 Parallel Algorithm

The main idea of the parallel algorithm is to first find the set of nodes $V(\overline{T}^{\omega}_{e})$ disconnected when each tree edge e is removed. Then each nontree edge has an $n \times n$ matrix of inter vertex replacement paths computed. Each element of this $n \times n$ matrix is propagated up the transmuter structure. We keep only the minimum value computed so far for each element of the matrix. Once the propagating stage is complete, we use a parallel version of either procedure Post-MVE1 or precedure Post-MVE2 to solve problem MVE1 or MVE2, respectively.

```
Algorithm Parallel-MVE
   {*Input: the SP tree $T(z)$ rooted at $z$ for each $z \in V$. Output: the single MVE with respect to the
   all pairs shortest paths.*}
1. Compute $V(\overline{T}^{\omega}_{e})$ for each tree edge $e \in E$ in parallel using a parallel version of procedure Pre-MVE;
2. Compute the $n \times n$ matrix $A(f)$ for each nontree edge $f$;
3. Using the path labeling technique from Ref. [15] and the construction in Section 62.3, compute the
   $|V|$ minimum labels for each tree edge in parallel:
   3.1 For each nontree edge $(x, y)$, project its $n^2$ labels to every tree edge on the cycle $T^{\omega} \bigcup \{(x, y)\}$
       for each $\omega \in V$;
   3.2 For each tree edge, find the $n^2$ minimum labels among all labels assigned to it in Step 3.1;
4. Call a parallel version of either procedure Post-MVE1 or Post-MVE2 to calculate MVE1 or MVE2,
   respectively.
```

62.3.2.2.1 CRCW Implementation

The model of computation for the algorithm is the MINIMUM-CRCW PRAM. On this model, if a write conflict arises, the processor holding the minimum value is allowed to write. This processor writes its value if and only if the value to be written is smaller than any previously computed value.

Given a MINIMUM-CRCW PRAM with m processors, we can implement Tarjan's Shortest Path Sensitivity Analysis algorithm in $O(\log n)$ [34] time by a path labeling technique described in Ref. [15] and later used in Ref. [17]. This combines the techniques of Euler tour and list ranking using pointer jumping [35] to find the nearest common ancestor of a nontree edge (x, y) and assign the correct label to each edge on the paths from x and y to that ancestor. The Euler tour technique is used to split the path $P(x, y)$ into two subpaths by finding the lowest common ancestor of vertices x and y. We then use list ranking with pointer jumping to propagate the labels along the two subpaths.

In Step 1 of the parallel algorithm, the set $V(\overline{T}_e^{\omega})$ is calculated for each edge $e \in T^{\omega}$, for each $\omega \in V$, by applying the Euler tour technique and Lemma 62.1 as follows:

```
1.1 Root $T^{\omega}$ at $\omega$ using the Euler tour technique (see Ref. [35, Section 3.2])
1.2 for each $v \in V$ do
        for each $w \in V \setminus \{\omega\}$ do
            if $nca(v, w) = w$ then
                Set position $v$ at edge $(parent(w), w)$ to a 1.
```

Rooting a tree using the Euler tour technique requires $O(\log n)$ time on n EREW processors. There are n vertices, each of which requires n nearest common ancestor calculations. An edge can appear in at most n SP trees. None of the above steps requires concurrent read or write; thus, we can implement Step 1 in $O(\log n)$ time using n^3 EREW processors.

Step 2 can be done in constant time on mn^2 processors as there are at most $m - n + 1$ nontree edges, and each nontree edge requires n^2 constant-time path calculations. Step 3 can be done in $O(\log n)$ time using mn^2 processors (for details see Ref. [25]).

If we calculate MVE1, procedure Post-MVE1 can be done in constant time on mn^2 processors as there are at most m distinct SP tree edges, and we calculate the minimum of at most n^2 elements for each of these. For procedure Post-MVE2, we compute the sum of at most n^2 elements for each tree edge. There are at most m such tree edges. Hence, this step can be done in $O(\log n)$ time using mn^2 processors.

62.4 Concluding Remarks

Identifying critical sections in a network for protection to avoid sudden system breakdown and disruption is a topic of increasing importance for network-centric computing and applications. In this chapter, we summarized the results made in our previous work on the problems of computing the MVEs with respect to MST and SP. For the first problem, we shown have approaches to develop both deterministic and randomized solutions for computing k-MVE and their parallel executions. For the second problem, we described efficient sequential and parallel solutions using an auxiliary graph transmuter [34]. Though the contents of this chapter may not cover all research activities on this topic, we hope that they can provide some insight to understanding the approaches and techniques for computing MVE with respect to certain properties in a graph. These may also be helpful to devising solutions for other relevant problems.

References

[1] Lobore, S. H., Ratliff, H. D., and Sicilia, G. T., Determining the most vital link in a flow network, *Naval Res. Logist. Quart.*, 18, 497, 1971.

[2] Wollmer, R., Removing arcs from a network, *Oper. Res.*, 12, 934, 1965.

[3] Ratliff, H. D., Lobore, S. H., and Sicilia, G. T., Finding the n most vital links in flow networks, *Manage. Sci.*, 21, 531, 1975.

[4] Corley, H. W. and Sha, D. Y., Most vital links and nodes in weighted networks, *Oper. Res. Lett.*, 1, 157, 1982.

[5] Malik, K., Mittal, A. K., and Gupta, S. K., The k most vital edges in the shortest path problem, *Oper. Res. Lett.,* 8, 223, 1989.

[6] Malik, K., Mittal, A. K., and Gupta, S. K., Erratum: the k most vital arcs in the shortest path problem, *Oper. Res. Lett.*, 9, 283, 1990.

[7] Ball, M. O., Golden, B. L., and Vohra, R. V., Finding the most vital arcs in a network, *Oper. Res. Lett.*, 8, 73, 1989.

[8] Hsu, L. H., Jan, R. H., Lee, Y. C., Hung, C. N., and Chern, M. S., Finding the most vital edge with respect to minimum spanning tree in weighted graphs, *Inf. Proc. Lett.*, 39, 277, 1991.

[9] Lin, K.-C. and Chern, M. S., The most vital edges in the minimum spanning tree problem, *Inf. Proc. Lett.*, 45, 25, 1993.

[10] Bhosle, A. M. and Gonzalez, T. F., Algorithms for single link failure recovery and related problems, *J. Graph Alg. Appl.*, 8(3), 275, 2004.

[11] Bhosle, A. M., Improved algorithms for replacement paths problems in restricted graphs, *Oper. Res. Lett.*, 33(5), 459, 2005.

[12] Hershberger, J. E., Suri, S., and Bhosle, A. M., On the difficulty of some shortest path problems, *Proc. STACS,* 2003, p. 343.

[13] Hsu, L. H., Wang, P. F., and Wu, C. T., Parallel algorithms for finding the most vital edge with respect to minimum spanning tree, *Parallel Comput.*, 18, 1143, 1992.

[14] Iwano, K. and Katoh, N., Efficient algorithms for finding the most vital edge of a minimum spanning tree, *Inf. Proc. Lett.*, 48, 211, 1993.

[15] Katajainen, J. and Träff, J. L., Simple parallel algorithms for the replacement edge problem and related problems on minimum spanning trees, TR DIKU-94/18, Department of Computer Science, University of Copenhagen, 1994.

[16] Liang, W. and Shen, X., Finding the *k* most vital edges in the minimum spanning tree problem, *Parallel Comput.,* 23(13), 1889, 1997.

[17] Shen, H., Improved parallel algorithms for finding the most vital edge of a graph with respect to minimum spanning tree, *Int. J. Comput. Math.*, 75(2), 129, 2000.

[18] Suraweera, F. and Maheshwari, P., A parallel algorithm for the most vital edge problem on the CRCW-SIMD computational model, *Proc. Computer Science Conf.*, 1994.

[19] Suraweera, F., Maheshwari, P., and Bhattacharya, P., Optimal algorithms to find the most vital edge of a minimum spanning tree, TR CIT-95-21, School of Comput. and Inf. Tech., Griffith University, 1995.

[20] Shen, H., Finding the *k* most vital edges with respect to minimum spanning tree, *Acta Inf.*, 36, 405, 1999.

[21] Frederickson, G. N. and Solis-Oba, R., Increasing the weight of minimum spanning trees, *Proc. SODA,* 39, 1996, p. 1.

[22] Shen, H., On exact algorithms for finding the *k* most vital edge with respect to minimum spanning tree, *Proc. Parallel and Dist. Comput. and Sys.*, 1996, p. 446.

[23] Fredman, M. L. and Tarjan, R. E., Fibonacci heaps and their uses in improved network optimization algorithms, *Proc, FOCS,* 1984, p. 338.

[24] Bar-Noy, A., Khuller, S., and Schieber, B., The complexity of finding most vital arcs and nodes, TR CS-TR-3539, University of Maryland Inst. for Advanced Studies, 1995.

[25] Venema, S., Shen, H., and Suraweera, F., NC algorithms for the single most vital edge problem with respect to shortest paths, *Inf. Proc. Lett.,* 60(5), 243, 1996.

[26] Venema, S., Shen, H., and Suraweera, F., *NC* algorithms for the single most vital edge problem with respect to all pairs shortest paths, *Parallel Proc. Lett.*, 10(1), 51, 2000.

[27] Nardelli, E., Proietti, G., and Widmayer, P., Finding the detour-critical edge of a shortest path between two nodes, *Inf. Proc. Lett.*, 67(1), 51, 1998.

[28] Nardelli, E., Proietti, G., and Widmayer, P., A faster computation of the most vital edge of a shortest path, *Inf. Proc. Lett.*, 79(2), 81, 2001.

[29] Nagamochi, H. and Ibaraki, T., Linear time algorithm for finding *k*-edge-connected and *k*-vertex-connected spanning subgraphs, *Algorithmica*, 7, 583, 1992.

[30] Katoh, N., Ibaraki, T., and Mine, H., An algorithm for finding *k* minimum spanning trees, *SIAM J. Comput.*, 10, 247, 1981.

[31] Cole, R., Klein, P. N., and Tarjan, R. E., A linear-work parallel algorithm for finding minimum spanning trees, *Proc SPAA,* 1994.

[32] Chong, K. W. and Lam, T. W., Finding minimum spanning trees on the EREW PRAM, manuscript, 1996.

[33] Knott, G. D., A numbering system for combinations, *CACM*, 17, 45, 1974.

[34] Tarjan, R. E., Sensitivity analysis of minimum spanning trees and shortest path trees, *Inf. Proc. Lett.*, 14, 30, 1982.

[35] Jájá, J., *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, MA, 1992.

[36] Ho, J., Lee, D. T., Chang, C., and Wong, C. K., Minimum diameter spanning trees and related problems, *SIAM J. Comput.*, 20(5), 987, 1991.

[37] Brown, W. G., *Reviews in Graph Theory,* AMS, Providence, RI, 1980.

63

Stochastic Local Search Algorithms for the Graph Coloring Problem

Marco Chiarandini
University of Southern Denmark

Irina Dumitrescu
HEC Montreal

Thomas Stützle
Free University of Brussels

63.1 Introduction

The graph (vertex) coloring problem (GCP) is a central problem in graph theory [1]. It consists of finding an assignment of colors to vertices of a graph in such a way that no adjacent vertices receive the same color. Graph coloring problems arise in many real-life applications such as register allocation [2], air traffic flow management [3], frequency assignment [4], light wavelengths assignment in optical networks [5], and timetabling [6,7].

In the GCP, one is given an undirected graph $G = (V, E)$, with V being the set of $|V| = n$ vertices and E the set of edges. A *k-coloring* of G is a mapping $\varphi : V \mapsto \Gamma$, where $\Gamma = \{1, 2, \ldots, k\}$ is a set of $|\Gamma| = k$ integers, each one representing a color. A k-coloring is *feasible* or *proper* if for all $[u, v] \in E$ it holds that $\varphi(u) \neq \varphi(v)$; otherwise it is *infeasible.* If for some $[u, v] \in E$ it is $\varphi(u) = \varphi(v)$, the vertices u and v are *in conflict.* The *conflict set* V^c is the set of all vertices that are in conflict. A k-coloring can also be seen as a partitioning of the set of vertices into k disjoint sets, called *color classes*, and represented as a partitioning of V, $C = \{C_1, \ldots, C_k\}$. Finally, if some vertices are assigned to color classes while others are not, the coloring is said to be a *partial coloring.*

The GCP can be posed as a decision or an optimisation problem. In the decision version, also called the *(vertex) k-coloring problem*, the question to be answered is whether for some given k a feasible k-coloring exists. The optimisation version of the GCP asks for the smallest number k such that a feasible k-coloring exists; for a graph G, this number is called the *chromatic number* χ_G. The problem of finding the chromatic number is often approached by solving a sequence of k-coloring problems: an initial value of k is considered and each time a feasible k-coloring is found, the value of k is decreased by one. The chromatic number is found when for some k the answer to the decision version is "no" i.e., a feasible k-coloring does not exist. In this case, $\chi_G = k+1$. If a feasible coloring cannot be found but no proof of its nonexistence is given, as it is typically the case with heuristic algorithms, $k+1$ is an upper bound on the chromatic number.

It is well known that the k-coloring problem for general graphs is $\mathcal{NP}$-complete and that the chromatic number problem is $\mathcal{NP}$-hard [8]; only for a few special cases polynomial-time algorithms are known. In some sense, the GCP is among the hardest problems in $\mathcal{NP}$, since approximation ratios of $n^{1-\epsilon}$ cannot be achieved in polynomial time, unless $\mathcal{NP} = \mathcal{ZPP}$ [9]; i.e., it is very unlikely to find polynomial-time algorithms that guarantee a constant approximation ratio. The best absolute performance guarantee of $\mathcal{O}(n(\log\log n)^2/\log^3 n)$ is given for an approximation algorithm presented in Ref. [10]. More details on the approximability of graph coloring can be found in Ref. [11].

Due to its importance, many attempts have been made to tackle the GCP algorithmically. Various exact algorithms, including specialised branch-and-bound algorithms [12,13] or approaches based on general integer programming formulations of the GCP have been tested [14–16]. Probably the best known exact algorithm is Brélaz' modification of Randall–Brown's coloring algorithm [12]. While exact algorithms can be very effective on specific classes of graphs, their performance for many large graphs is often rather poor [16,17]. Therefore, a significant amount of research has focused on stochastic local search (SLS) algorithms for the GCP (see Chapter 19 for an overview of SLS methods). The available SLS approaches range from rather simple but very fast construction methods over iterative improvement algorithms to sometimes rather complex SLS algorithms, which are currently the best performing approximate algorithms for many classes of GCP instances. This chapter is an overview of available SLS algorithms and reports some indicative comparison of the performance of several such algorithms.

63.2 Benchmark Instances

The available benchmark instances for the GCP are either graphs randomly generated or graphs derived from some practically relevant application. The most frequently used benchmark instances are available from the web page *COLOR02/03/04: Graph Coloring and its Generalizations* at `http://mat.tepper.cmu.edu/COLOR04` and an earlier DIMACS challenge [18]. Next, some instance classes are described which will be used in the experimental analysis presented in Section 63.7.

Uniform random graphs. This class comprises graphs of a variable number of vertices, n, where each of the $n(n-1)/2$ possible edges is generated independently at random with probability p. The instance have identifiers DSJC$n.p$, with $n \in \{125, 250, 500, 1000\}$ and $p \in \{0.1, 0.5, 0.9\}$. They stem from one of the first thorough experimental studies of SLS algorithms for the GCP [19]. For these graphs, probabilistic estimates of the chromatic number exist [20–22].

Flat graphs. These graphs are random graphs generated according to an equipartitioning of vertices into k sets. Edges are then generated to respect some constraints on the resulting degrees of the vertices until a given edge density is obtained [23]. The value of k is an upper bound on the chromatic number of the graph. These instances are denoted as flatn_k, with $n = 300$; $k \in \{20, 26, 28\}$ and $n = 1000$; $k \in \{50, 60, 76\}$.

Leighton graphs. Leighton graphs are random graphs of density below 0.25, which are constructed by first partitioning vertices into k distinct sets representing the color classes and then assigning edges only between vertices that belong to different sets. The chromatic number is guaranteed to be k by implanting cliques of sizes ranging from 2 to k into the graph. The graphs are denoted as le450_kx, where 450 is the number of vertices, k the chromatic number of the graph, and $x \in \{a, b, c, d\}$ a letter used to distinguish different graphs with the same characteristics. Graphs with letters c and d have higher edge density than those with a and b.

Quasigroup graphs. A Quasigroup is an algebraic structure on a set with a binary operator. The constraints on this structure define a Latin square, that is a square matrix with elements such that entries in each row and column are distinct. A Latin square of order n has n^2 vertices and $n^2(n-1)$ edges, corresponding to $2n$ cliques, a clique per row/column, each of size n. The chromatic number of Latin square graphs is n.

Latin square graphs originated by Quasigroups are denoted by qg.order$n$. Latin square graphs also arise in experimental design for statistical analysis. The instance latin_square_10, which has an unknown chromatic number, is one such example with preassigned experiments [24].

Queens graphs. The n-queens problem asks whether it is possible to place n queens on a $n \times n$ grid such that no pair of queens is in the same row, column, or diagonal. This problem can be posed as a GCP and a feasible solution with n queens exists if, and only if, the resulting queen graphs have a feasible coloring with n colors. Queen graphs are highly structured instances and their edge density decreases with their size; they are denoted by queen$n$_n.

WAP graphs. These graphs arise in the design of transparent optical networks [5] and are denoted by wap0ma, where $m = \{1, \ldots, 8\}$. They have between 905 and 5231 vertices. All instances have a clique of size 40.

Jacobian estimation graphs. These graphs stem from a matrix partitioning problem in the segmented columns approach to determine sparse Jacobian matrices [25]. They range in size from 662 to 1916 vertices and they are identified with the following names abb313GPIA, ash331GPIA, ash608GPIA, ash958GPIA, introduce more space between the two paragraphs and will199GPIA.

The DIMACS benchmark repository contains some other instances which were identified as *easy*. For these instances, some combination of preprocessing rules and simple construction heuristics, which are introduced in the next two sections, suffice to find a coloring with the known chromatic number. Forty-five instances were identified as easy. They are the Mycielski graphs, the graphs from register allocation for variables in compiled code [24], graphs from Knuth's Stanford GraphBase [26], and the almost 3-colorable graphs with embedded four cliques [27]. Two further classes of graphs that are a generalisation of Mycielski graphs (the insertions and full insertions graphs due to Caramia, M. and Dell'Olmo P., personal communication) and graphs for course scheduling [24] are not useful to determine differences among algorithms.

63.3 Preprocessing and Heuristic Reduction Rules

In the chromatic number problem, preprocessing can be applied to reduce a graph G to a graph G' such that a feasible k-coloring for G can be derived by construction rules from any feasible k-coloring of G'. Next, the two preprocessing rules presented in Ref. [28] are given.

Rule 1. Remove all vertices in G that have a degree less than the size of the largest known clique $\widehat{\omega}(G)$. Knowing that the degree of a vertex u is less than $\widehat{\omega}(G)$ guarantees that at least one color that is not used in the set of adjacent vertices can be assigned to u without breaking feasibility. (This rule can be applied when solving the k-coloring problem by replacing $\widehat{\omega}(G)$ by k.)

Rule 2. Remove any vertex $v \in V$ for which there is a $u \in V$, $v \neq u$ and $[u, v] \notin E$, such that u is adjacent to every vertex to which v is adjacent (subsumption). In this case, any color that can be assigned introduce more space between the two paragraphs to u can also be assigned to v.

These two rules can be applied in any order and if one rule applies, it may make possible further reductions by the other rule. Hence, in the preprocessing stage, the two rules are applied iteratively until no vertex can be removed anymore. The rules are easy to apply. Rule 1 requires $\mathcal{O}(|V|)$ operations once a clique has been found heuristically and the degree of each vertex is known. Rule 2 is more costly and its time complexity is $\mathcal{O}(|V|^3)$. The overall reduction time is, however, insignificant in practice. Among the challenging instances considered, preprocessing is effective only for the WAP instances.

The graph can also be reduced heuristically. One such procedure consists of reducing the graph by removing maximal independent sets [29–31]. Typically, this procedure is accomplished with the companion strategy of minimising the number of edges in the residual graph. In contrast to the previously mentioned preprocessing rules, this procedure may make it impossible to find the chromatic number of the original graph because it is a priori unknown how the independent sets should be constructed such that an optimal coloring is obtained from a coloring of the residual graph and the coloring for the independent sets. Nevertheless, for some graphs such a heuristic procedure has contributed significantly to the improvement of the solution quality for SLS algorithms [29,30,32,33].

63.4 Construction Heuristics

The fastest methods to generate a feasible coloring are construction heuristics. Most of these heuristics for the GCP belong to the class of *sequential* heuristics that start from a set of empty color classes $\{C_1, \ldots, C_k\}$, where $k = |V|$, and then iteratively add vertices to color classes until a complete coloring is reached. Each iteration of the heuristic consists of two steps: first, the next vertex to be colored is chosen, and then this vertex is assigned to a color class. The order in which the vertices are colored corresponds to a permutation π of the vertex indices that can be determined either *statically* before assigning the colors, or *dynamically* by taking into account the partial coloring for the choice of the next vertex. The choice of the color class at each construction step is typically based on the *greedy heuristic* that adds at iteration i the vertex $\pi(i)$ to the color class with the lowest possible index such that the partial coloring obtained after coloring vertex $\pi(i)$ remains feasible, therefore, trying to minimise the number of nonempty color classes.

Clearly, the result of the greedy heuristic depends on the permutation π. The simplest way to derive π in a static way is by using the *random order sequential* (ROS) *heuristic*, which simply generates a random permutation. Several other ways for generating π statically exist, like largest degree first, smallest degree last, etc. However, static sequential methods are typically dominated by dynamic ones [21]. Probably, the most widely known dynamic heuristic is the DSATUR *heuristic* that is derived from the exact DSATUR algorithm of Brélaz [12]. In DSATUR, the vertices are first arranged in decreasing order of their degrees and a vertex of maximal degree is inserted into C_1. Then, at each construction step the next vertex to be inserted is chosen according to the *saturation degree*, i.e., the number of differently colored adjacent vertices. The vertex with the maximal saturation degree is chosen and inserted according to the greedy heuristic. Ties are broken preferring vertices with the maximal number of adjacent, still uncolored vertices; if further ties remain, they are broken randomly. Other dynamic heuristics for determining π were studied in Refs. [34,35].

A different strategy for generating colorings is to iteratively extract independent sets from the graph. The most famous such heuristic is the *recursive largest first* (RLF) heuristic proposed by Leighton [7]. RLF iteratively constructs a color class C_i by first assigning to it a vertex v with maximal degree from the set V' of still uncolored vertices; initially it is $V' = V$. Next, all the vertices in V' that are adjacent to v are removed from V' and inserted into a set U, which is initially empty; U is the set of vertices that cannot be added to color class C_i anymore. Then, while V' is not empty, the vertex $v' \in V'$ that has the largest number of edges $[v', u]$, with $u \in U$, is chosen; v' is added to C_i, removed from V', and all vertices in V' adjacent to v' are moved into U. Ties are broken, if possible, choosing the vertex that has minimal degree in V', otherwise randomly. These steps are iterated until V' is empty and the same steps are repeated with the residual graph consisting of the vertices in U.

We have compared experimentally the three construction heuristics DSATUR, RLF, and ROS [17]. Their behaviour is stochastic because random choices are used to break ties, hence each of the algorithms was run 10 times on all 125 instances of the DIMACS benchmark sets as of end 2004. The conclusion was that RLF performs statistically significantly better than DSATUR for most instance classes and both heuristics are by a large margin better than ROS. RLF is not significantly better than DSATUR only on the insertions, full insertions, and course scheduling graphs. With respect to computation time, although RLF and DSATUR have the same time complexity $\mathcal{O}(|V|^3)$, RLF is in practice much more time-consuming. ROS with a complexity of $\mathcal{O}(|V|^2)$ is the fastest. For example, on the largest instance tested (10,000 vertices), RLF takes on average ~18.5 s to color the graph with 101 colors, DSATUR 3 s to color it with 103 colors, and ROS < 1 s to produce a coloring with 106 colors (the computer used was a 2 GHz AMD Athlon MP 2400+ Processor with 256 KB cache and 1 GB of RAM memory). A more detailed analysis showed that the computation time of RLF depends not only on the instance size, as it is the case for DSATUR, but it is also affected by the graph density and the final number of colors used [17].

Note that the construction heuristics described above can also be used if a fixed number $k < |V|$ of color classes is available. In that case, the usual steps of the construction algorithms are followed as long as the number of used color classes remains below k. Then, if a vertex cannot be colored legally, it is added to a color class randomly or according to some other heuristic that tries to minimise the number of arising conflicts. The result of these so modified construction heuristics is typically an infeasible k-coloring, which can serve as an initial solution for improvement methods.

63.5 Neighbourhoods for Local Search

Once an initial (feasible or infeasible) coloring is generated by construction algorithms, one may try to improve it through the application of iterative improvement algorithms. These algorithms may be used within two different approaches to solve the GCP: as a sequence of decision problems or directly as an optimisation problem. The first approach corresponds to leaving the numbers of colors *fixed* to some value k at each stage. Subsequently, once a feasible coloring with k colors is found, the number of colors is reduced by one. The second approach allows the number of used colors to *vary* throughout one single trial of the local search algorithm. To apply iterative improvement algorithms one needs to define the set of candidate solutions, a neighbourhood relation, and an evaluation function. Two choices are possible for the set of *candidate solutions*: one may opt for the algorithm to work on *complete colorings*, where each candidate solution represents a possibly infeasible partitioning of the set of vertices into color classes, or to work on *partial colorings*, where candidate solutions are also those with a subset of the vertices left without any color assigned. Next, several neighbourhood relations and evaluation functions of candidate solutions are introduced, differentiating among possible choices of how to tackle the GCP.

63.5.1 Strategy 1: *k* Fixed, Complete Colorings

This approach works on a partitioning of V into k color classes: $C = \{C_1, \ldots, C_k\}$. The most widely used evaluation function counts the number of edges that have their end points in the same color class. Formally, the function can be described as $g(C) = \sum_{i=1}^{k} |E_i|$, where E_i is the set of edges with both end points in C_i. A candidate solution with an evaluation function value of zero corresponds to a proper k-coloring. An alternative evaluation function would be $g(C) = |V^C|$. However, this function is much less used, possibly because it would lead to a large number of neighbours with the same evaluation function value, therefore inducing large plateaus in the search landscape.

1-exchange neighbourhood. The most frequently used neighbourhood structure in this setting is the 1-exchange neighbourhood, in which two colorings are neighbours if they differ in the color class assignment of exactly one vertex. That is, to obtain a neighbour of C, one vertex u is moved from a color class C_j, $j \in \{1, \ldots, k\}$, into a different color class C_l, $l \neq j$. Often, the 1-exchange neighbourhood is further restricted to change only the color class assignment of vertices that are in conflict, since only these modifications can lead to a decrease of the evaluation function; this is called the *restricted 1-exchange neighbourhood.*

Swap neighbourhood. In the swap neighbourhood, exactly one vertex $v \in V^c$ exchanges the color class with another vertex $u \in V$. This neighbourhood is of quadratic size and it is used only very rarely.

Cyclic exchange and path exchange neighbourhoods. An extension of the 1-exchange and swap neighbourhoods are the path and cyclic exchange neighbourhoods (see Figure 63.1). Both the cyclic and the path exchange are sequences of 1-exchanges. A *cyclic exchange* of length m acts on a sequence of distinctly colored vertices $(u_1, \ldots, u_m)$. For simplicity, the color class of any u_i, $i = 1, \ldots, m$, will be denoted by C_i. The cyclic exchange moves any u_i, $i = 1, \ldots, m$, from C_i into C_{i+1}. For convention, $C_{m+1} = C_1$. A cyclic exchange does not change the cardinality of the color classes involved in the move. In a *path*

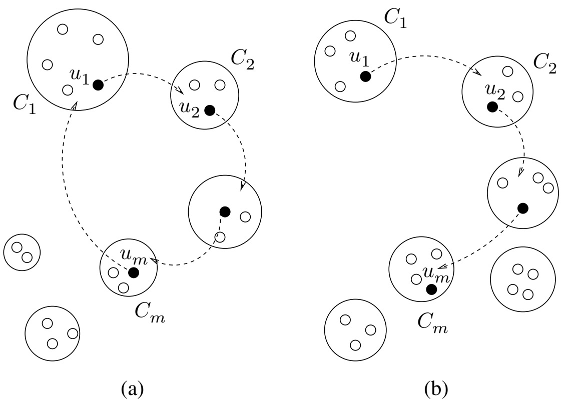


FIGURE 63.1 The black vertices are the vertices involved in the cyclic and path exchange: (a) cyclic exchange, (b) path exchange.

exchange instead, u_m remains in C_m. Hence, the sequence of exchanges is not closed and the cardinality of C_1 and C_m is modified. The cyclic and path exchange neighbourhoods are examples of very large-scale neighbourhoods (VLSN). The problem of finding a good neighbour within these neighbourhoods can be reduced to searching the least cost cycle or cost path in an improvement graph. If these problems can be solved exactly, then the best neighbour is found. Refer to Chapter 20 for more details on how an improvement graph is built and can be searched exactly and heuristically. Details on the implementation of a VLSN for the GCP can be found in Ref. [17].

Other neighbourhoods. Similar to the path exchange neighbourhood is the ejection-chain neighbourhood defined in Ref. [36], where the length of the sequences was limited to 3. Other neighbourhoods, rather fancy at times, were proposed in the context of a variable neighbourhood approach [37]. However, the overall contribution of these neighbourhoods is somewhat unclear, especially when they are used inside a more complex SLS algorithm.

63.5.2 Strategy 2: *k* Fixed, Partial Colorings

This approach works on a partitioning of V into $k+1$ color classes: $C = \{C_1, \ldots, C_k, C_{imp}\}$. The partial coloring $\bar{C} = \{C_1, \ldots, C_k\}$ is usually required to be feasible. The color class C_{imp} is called the *impasse* class [33] and it contains the "uncolored" vertices. The goal of the local search is to try to empty color class C_{imp}, while maintaining the partial coloring $\bar{C}$ feasible. The most widely used evaluation function in this case is $g(C) = \sum_{v \in C_{imp}} d(v)$, where $d(v)$ is the degree of vertex v. Analogously to complete colorings, an alternative would be to minimise $g(C) = |C_{imp}|$. However, this appears to lead to worse performance [31].

i-swap neighbourhood. A neighbour of C in the i-swap neighbourhood is obtained by moving a vertex v from C_{imp} into another color class C_i, followed by moving all vertices of C_i that share an edge with v into C_{imp} so that the partial coloring $\bar{C}$ remains feasible [33].

Other neighbourhoods. The neighbourhoods discussed in the section dedicated to Strategy 1 could, at least in principle, also be applied for algorithms using strategy 2; however, it would probably be difficult to maintain the partial coloring feasible and additional penalties for infeasibility may be required in the evaluation function. So far, such an approach appears not to have been tried.

63.5.3 Strategy 3: *k* Variable, Complete Colorings

The final strategy discussed allows the number of color classes to vary during the local search. In almost all these approaches, the current candidate coloring is forced to be complete and feasible. The local search

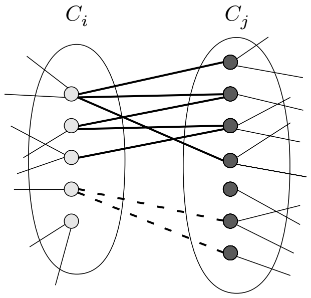


FIGURE 63.2 A Kempe chain between color classes C_i and C_j is indicated by the thick edges.

then tries to minimise the number of color classes. Hence, the simplest evaluation function would be to count the number of colors currently used; however, since moves that remove one color class completely will be certainly very rare, the guidance provided by this evaluation function would be minor. As an alternative, Johnson et al. [19] proposed to use $g(C) = -\sum_{i=1}^{k}(|C_i|)^2$, which biases the search towards a small number of color classes. The most widely used neighbourhood structure for this strategy is based on Kempe chains.

Kempe chains neighbourhood. A *Kempe chain* K is a set of vertices that form a component (a maximal connected subgraph) in the subgraph G' of G induced by the vertices that belong to two (disjoint) color classes C_i and C_j of C. A Kempe chain exchange applied to a feasible coloring produces again a feasible coloring (a *Kempe chain neighbour*) by moving all vertices of C_i that belong to the Kempe chain K into the color class C_j and vice versa. An example of a Kempe chain is given in Figure 63.2.

An alternative to enforcing coloring feasibility is to allow also infeasible colorings. In that case, the same neighbourhoods as under the k fixed strategy can be applied, but the evaluation function needs now also to guide the search towards feasible candidate solutions. One such evaluation function is $g'(C) = -\sum_{i=1}^{k}|C_i|^2 + \sum_{i=1}^{k} 2|C_i||E_i|$, where E_i is the set of vertices with both end points in C_i [19]. The second term in g' is used to penalise conflicts between adjacent vertices.

Finally, note that no approach that leaves k variable and uses partial colorings is known, although such an approach would be conceivable.

63.5.4 Neighbourhood Examination and Data Structures

The various neighbourhoods that have been described can be restricted and explored in various ways. It is intuitively clear that neighbourhood restrictions are more important for large neighbourhoods than for the small ones. Nevertheless, computational results with SLS algorithms suggest that restrictions to moves that involve only vertices in V^c are essential even for the 1-exchange neighbourhood.

Regarding the neighbourhood search strategy, one can distinguish mainly between best- and first-improvement strategies. In best-improvement, the best neighbouring candidate solution (best w.r.t. neighbourhood restrictions, if these are used) is accepted, breaking ties randomly; in first-improvement, the first improving candidate solution found when scanning the neighbourhood replaces the current one. Somehow intermediate between these two is the strategy followed in the *min-conflicts heuristic* [38], which searches the restricted 1-exchange neighbourhood in a two-stage process. In a first stage, a vertex is chosen uniformly at random from the conflict set; in a second stage, this vertex is moved into a color class such that the number of conflicts is minimised, breaking ties randomly.

Finally, let us note that the local search algorithms for the GCP require the usage of appropriate data structures to support caching and updating strategies to make the evaluation of the neighbouring candidate solutions as efficient as possible. Typically, the use of elaborated data structures is more important for best-improvement local search algorithms and those making use of large neighbourhoods. We refer to Ref. [17] for a short discussion of efficient caching and updating strategies.

63.6 Stochastic Local Search Algorithms

In the case of the GCP, iterative improvement algorithms have received rather little attention, probably because of their generally poor performance when used as stand-alone algorithms. Much more attention has focused on the application of more complex SLS algorithms. In what follows, a selection of some of the best-performing SLS algorithms is presented, as well as a few recently developed ones. The selection was biased by the desire to cover different approaches and methods. Where appropriate, there also is a short overview of algorithms that are related to the ones described, covering in this way the majority of the SLS algorithms proposed so far for the GCP. If nothing else is said in the text, the evaluation functions used are the standard ones given for the various strategies described in the previous section.

63.6.1 Strategy 1: *k* Fixed, Complete Colorings

Tabu search with 1-exchange neighbourhood. Tabu search algorithms based on the 1-exchange neighbourhood ($\mathsf{TS}_{\text{1-ex}}$) are probably the most frequently applied SLS algorithms for the GCP (see Chapter 23 for details on tabu search). Such an algorithm was first proposed by Hertz and de Werra [30] and was later improved leading to the most performing tabu search variant to date by Dorne, Galinier, and Hao [39,40]. $\mathsf{TS}_{\text{1-ex}}$ chooses at each iteration a best nontabu or tabu but aspired neighbouring candidate solution from the restricted 1-exchange neighbourhood. If a 1-exchange move puts vertex v from color class C_i into C_j, it is forbidden to re-assign vertex v to C_i in the next tt steps; the tabu status of a neighbouring solution is overruled if it improves over the best candidate solution found so far (aspiration criterion). If more than one move produces the same effect on the evaluation function, one of those moves is selected uniformly at random. The tabu list length in $\mathsf{TS}_{\text{1-ex}}$ is set to $tt = \text{random}(A) + \delta \cdot |V^c|$, where $random(A)$ is an integer uniformly chosen from $\{0, \ldots, A\}$ and δ and A are parameters. Since tt depends on $|V^c|$, the tabu list length varies dynamically with the evaluation function value.

Tabu search with very large-scale neighbourhood. In this algorithm, a neighbourhood obtained from the composition of the path exchange and cyclic exchange neighbourhoods is used, referred to as the *cyclic and path exchange neighbourhood.* The embedding of this neighbourhood into a tabu search algorithm analogous to $\mathsf{TS}_{\text{1-ex}}$ results in an algorithm called $\mathsf{TS}_{\text{VLSN}}$ [17]. Several variants of $\mathsf{TS}_{\text{VLSN}}$ have been studied and the best performing one of these first selects the best nontabu move in the restricted 1-exchange neighbourhood as in $\mathsf{TS}_{\text{1-ex}}$. If this move is a plateau move, (i.e., the best neighbouring solution has the same evaluation function value as the current one), then the cyclic and path exchange neighbourhood is searched. In all other cases the 1-exchange move is applied and the tabu list is updated. The tabu mechanism is applied to the search for cyclic and path exchanges by discarding any neighbouring candidate solution that involves the reassignment of a vertex to some color class that is currently tabu. The tabu list is updated by considering the path or the cyclic exchange as a composition of 1-exchanges and the tabu duration tt for a specific vertex-color–class pair (v, i) is chosen using the rule of $\mathsf{TS}_{\text{1-ex}}$. Yet, contrarily to $\mathsf{TS}_{\text{1-ex}}$, $\mathsf{TS}_{\text{VLSN}}$ does *not* use an aspiration criterion.

Min-conflicts heuristic. One of the most effective extensions of the basic min-conflicts heuristic is a tabu search variant of it ($\mathsf{MC}\text{-}\mathsf{TS}_{\text{1-ex}}$) [41]. It uses the same two-stage selection process as the min-conflicts heuristic, which was described in the previous section, but in the second stage it only allows to move the vertex into a color class that is not tabu, analogous to $\mathsf{TS}_{\text{1-ex}}$. If all color classes are tabu, one is chosen randomly. One advantage of this neighbourhood examination strategy is that it does not require the usage of sophisticated caching and updating schemes as required, for example, by $\mathsf{TS}_{\text{1-ex}}$; hence, it allows for an easier implementation. In addition, the chances of cycling are reduced due to the random choices especially in the first stage of the selection process, allowing for shorter tabu lists.

Guided local search. Guided local search (GLS) is an SLS method that modifies the evaluation function to escape from local optima [42]. An application of GLS to the GCP was proposed in Ref. [17]. In this algorithm, GLS uses an augmented evaluation function g' defined as

$$g'(C) = g(C) + \lambda \cdot \sum_{i=1}^{|E|} w_i \cdot I_C(i)$$

where $g(C)$ is the usual evaluation function, λ a parameter that determines the influence of the penalties on the augmented cost function, w_i the penalty weight associated to edge i, and $I_C(i)$ an indicator function that takes the value 1 if the end points of edge i are in conflict in C and 0 otherwise. The penalties are initialised to 0 and are updated each time an iterative improvement algorithm reaches a local optimum of g'. The modification of the penalty weights is done by first computing a utility u_i for each violated edge, $u_i = I_C(i)/(1 + w_i)$, and then incrementing the penalties of all edges with maximal utility by one. The underlying local search is a best-improvement algorithm in the restricted 1-exchange neighbourhood. Once a local optimum is reached, the search continues for a maximum number of *sw* plateau moves before the evaluation function g' is updated.

Iterated local search. TS$_{\text{1-ex}}$ can be used as a local search inside hybrid SLS methods like iterated local search (ILS) [43]. In the ILS algorithm presented in Ref. [44], TS$_{\text{1-ex}}$ is run until the best solution found does not change for l_{LS} iterations. A perturbation is then applied to the best coloring found so far and TS$_{\text{1-ex}}$ is run again. In the perturbation, a number k_r, $k_r < k$, of color classes is randomly chosen and the color class membership of all vertices in those color classes is changed. The ROS heuristic bounded by k and with the further strong constraint of avoiding the reinsertion of a vertex into its previous color class is used to accomplish this task. The tabu list of TS$_{\text{1-ex}}$ is emptied before applying the perturbation, while the exchanges caused by the perturbation are inserted in the tabu list.

Other approaches that are based on the same or similar SLS methods have been studied: a predecessor of the ILS algorithm described above is presented in Ref. [45], an ILS algorithm that uses a permutation of the color classes in subgraphs as a perturbation is given in Ref. [46]. Similar in spirit is also the iterated greedy solution reconstruction [47].

Evolutionary algorithms. The first evolutionary algorithm (EA) for the GCP is reported in Ref. [48]. The most successful EAs are hybrid methods that use TS$_{\text{1-ex}}$ to improve candidate solutions [29,32,39,40,49]. Among them, the best results so far have been reported for the hybrid evolutionary algorithm (HEA) [40]. HEA starts with a population P of candidate solutions, which is initialised by using the DSATUR construction heuristic restricted to k colors, and then iteratively generates new candidate solutions by first recombining two members of the current population that are improved by local search. For the recombination, the greedy partition crossover (GPX) is used [40]. Starting with two candidate partitionings (parents) $C^1 = \{C_1^1, \ldots, C_k^1\}$ and $C^2 = \{C_1^2, \ldots, C_k^2\}$, GPX generates a candidate solution (offspring) by alternately selecting color classes of each parent. At step i of the crossover operator, $i = 1, \ldots, k$, GPX chooses a color class with maximal cardinality from parent C^1 (if i is odd) or from parent C^2 (if i is even). This color class will become color class C_i of the offspring. Once C_i is chosen, the vertices that belong to it are removed from both parents. The vertices that remain in C^1 and C^2 after step k are added to a color class of the child, which for each vertex is chosen uniformly at random. The new candidate partitioning returned by GPX is then improved by TS$_{\text{1-ex}}$, run for l_{LS} iterations, and it is inserted in the population P replacing the worse parent. The population is reinitialised if the average distance between colorings in the population falls below a threshold of 20. Responsible for the high performance reported for the algorithms appears to be mainly the GPX crossover operator [46].

The adaptive search algorithm of Galinier et al. [50] also makes use of the GPX crossover, but it does not further improve on the performance of HEA. Another EA was proposed in Ref. [51] and a scatter search algorithm was proposed in Ref. [52]; however, none of these algorithms appears to surpass the peak performance of HEA.

Other methods. A greedy randomised adaptive search procedure was proposed in Ref. [53]. It uses a randomisation of RLF for the candidate solution construction and an iterative improvement algorithm in the 1-exchange neighbourhood. The reported results for low-degree graphs appear to be good. Another

SLS method, ant colony optimisation (ACO), has also been applied to the GCP in Ref. [54] (see Chapter 26 for more details on ACO). In that approach, several ways of defining the heuristic information were studied; the computational results appear to be worse than state-of-the-art, however, no local search was used to improve candidate solutions.

63.6.2 Strategy 2: *k* Fixed, Partial Colorings

Distributed coloration neighbourhood search. The distributed coloration neighbourhood algorithm proposed by Morgenstern [33] can be seen as an ILS algorithm that uses a simulated annealing (SA) algorithm for the local search. The SA algorithm is based on the i-swap neighbourhood and is run for I iterations or until a certain solution quality threshold is passed. Upon termination of the SA algorithm, a perturbation is applied that is defined by a random *s-chain exchange* that moves from the current coloring configuration to a new one with the same solution quality. An s-chain exchange can be seen as a generalisation of the Kempe chain exchange. It is defined through a vertex v and an ordered sequence of nonempty color classes $C_1, \dots C_s$, where $v \in C_1$, all color classes are distinct, and $s \leq k$. From this sequence, a directed graph with vertex set $V' = C_1 \cup \cdots \cup C_s$ and arc set $A = \{(u, w) \mid (u, w) \in E,\ u \in C_i \text{ and } w \in C_{i+1}\}$ is derived (for convention, $C_{s+1} = C_1$). In an s-chain, each vertex that is reachable from v in the digraph (V', A) is moved from color class C_i to C_{i+1}. Note that an s-chain, where all vertices in V' are reachable, would simply correspond to a relabelling of the color classes and, hence, would result in the very same partitioning; therefore, such a *total* s-chain is to be avoided and the neighbourhood is restricted to non-total s-chains. Different ways of combining these two steps and including them into a distributed computational environment were studied in Ref. [33].

More recently, a tabu search algorithm based on the i-swap neighbourhood was proposed [55]. The tabu criterion in this algorithm forbids adding into C_i vertices adjacent to a vertex v that was moved into a color class C_i. This interdiction acts for the tt iterations successive to the move of v.

63.6.3 Strategy 3: *k* Variable, Complete Colorings

Simulated annealing with Kempe chain neighbourhood. SA was among the first SLS methods applied to the GCP [56,19]. A comprehensive study of three SA algorithms was presented in Ref. [19]; among these three variants, two work with the number of colors k variable. The more promising one of the two allows only feasible colorings and uses the Kempe chain neighbourhood ($\mathsf{SA}_{\mathsf{Kempe}}$), while the other allows infeasible colorings and uses the 1-exchange neighbourhood. $\mathsf{SA}_{\mathsf{Kempe}}$ uses the evaluation function $g(C) = -\sum_{i=1}^{k}(|C_i|)^2$ and starts from an initial partitioning generated by the ROS heuristic. At each iteration of $\mathsf{SA}_{\mathsf{Kempe}}$, a neighbouring solution is generated in three steps; first, a nonempty color class C_i, a vertex $v \in C_i$, and a nonempty color class C_j are chosen uniformly at random but avoiding that C_i and C_j form a full Kempe chain, which would result simply in a relabelling of the two color classes; second, the Kempe chain K_{ij} of color classes C_i and C_j that contains vertex v is determined; and third, the Kempe chain exchange is applied. The generated neighbouring candidate solution C' is always accepted if it improves over the current candidate solution C and otherwise it is accepted with a probability of $\exp((g(C) - g(C'))/T)$, where T is a parameter called temperature. The parameter T is modified according to a rather standard cooling schedule [19].

63.7 Experimental Comparison of Stochastic Local Search Algorithms

This section reports numerical results for the SLS algorithms that were described in detail in Section 63.6. For this purpose, the challenging benchmark instances described in Section 63.2, i.e., those that are not recognised as easy, are used; the benchmark instances were not treated by the preprocessing rules given in Section 63.3. The algorithms compared are $\mathsf{TS}_{\mathsf{1\text{-}ex}}$, $\mathsf{TS}_{\mathsf{VLSN}}$, $\mathsf{MC\text{-}TS}_{\mathsf{1\text{-}ex}}$, ILS, GLS, HEA, $\mathsf{SA}_{\mathsf{Kempe}}$, and

XRLF [19], an extension of RLF. All algorithms were implemented under the same environment, sharing the same data structures as much as reasonable. A specific experimental set-up is used which considers the GCP in its optimisation version and each algorithm started from the number of colors k^{RLF} returned by the RLF heuristic. When a feasible k-coloring is found, a new coloring with $k - 1$ colors is created by uncoloring the vertices assigned to one selected color and recoloring each of the vertices by randomly choosing any of the remaining colors. Finally, all algorithms were allowed the same maximum computation time t_{max}, which is instance dependent, after preliminary experiments, it was decided to use TS$_{1\text{-ex}}$ as the reference algorithm and to set t_{max} to the average time TS$_{1\text{-ex}}$ needs to perform $I_{max} = 10^4 \times |V|$ iterations (averages are taken across 10 trials per instance). For each of the algorithms, there were attempts to link parameter settings to instance features where possible and set the remaining ones to some constant value that resulted in good performance across the whole benchmark set. Thus, the results presented below give rather an indication of the algorithms' robustness than necessarily their true peak performance.

Each of the algorithms was run 10 times on each instance. For each trial the minimal number of colors found by the algorithm was measured and the resulting data analysed using rank-based statistical methods. In particular, the measured results on each instance are transformed into a rank value in $1, \ldots, 80$ (eight algorithms are compared and each algorithm is run 10 times) and then the ranks within the instance classes described in Section 63.2 are aggregated. Note that applying the statistical tests to each of the instance classes separately avoids the incorrect bias towards problem classes that comprise more instances and it may help in distinguishing the particular strength or weakness of the algorithms for the various instance classes.

In Figure 63.3, we report for each instance class the results of the statistical analysis by the nonparametric rank-based Friedman test for an all-pairwise comparison [57]. The graphs are obtained by attaching error bars to a scatter plot of the estimated average rank versus algorithm labels. The length of the bars is adjusted so that the average rank of a pair of algorithms can be inferred to be different with statistical significance at the level of $\alpha = 0.05$ if their bars do not overlap. Numerical results on the performance of the algorithms are given in Table 63.1.

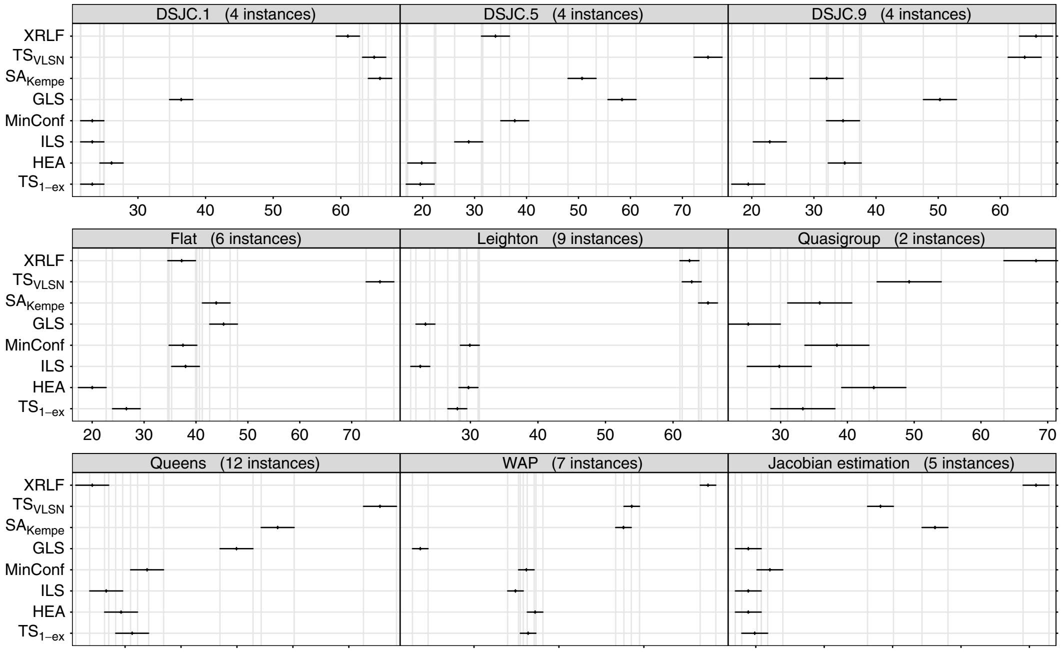


FIGURE 63.3 All-pairwise comparisons through confidence intervals for the SLS algorithms discussed in the text. The x-axis gives the average rank for the SLS algorithms.

TABLE 63.1 Numerical Results on the DIMACS Graph Coloring Instances.

	Bench.	time	$TS_{1\text{-ex}}$			HEA			ILS			$MC\text{-}TS_{1\text{-ex}}$			GLS			SA_{Kempe}			TS_{VLSN}			XRLF		
Instance	$(\chi, \widehat{\chi}^{best})$	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.
DSJC125.1	(–,5)	10	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	6	6	0	5	6	0	5	6	0
DSJC250.1	(–,8)	30	8	8	0.1	8	8	0	8	8	0.2	8	8	0.3	8	9	0	9	9	0.2	9	9	0	9	9	24.4
DSJC500.1	(–,12)	37	13	13	0.1	13	13	0.1	13	13	0.1	13	13	0.2	13	13	0.2	14	14	1.6	14	14	4.5	14	14	29.9
DSJC1000.1	(–,20)	174	21	21	2.8	21	21	164.3	21	21	5.9	21	21	10.4	21	22	0.8	23	23	46.7	23	23	90.5	22	22	169.9
DSJC125.5	(–,17)	14	17	17	0.5	17	17	1.3	17	17	1.6	17	17	6.6	18	18	0	18	18	0.4	18	19	2.1	18	18	2.5
DSJC250.5	(–,22)	47	28	28	22.3	28	28	30.8	28	28	33.6	29	29	2.8	29	30	0.9	29	30	2.7	32	32	6.2	29	30	5.4
DSJC500.5	(–,48)	168	49	50	35	50	50	100.3	50	50	105.8	50	51	20.4	52	52	81.4	51	51	47.3	55	55	138.5	50	50	123
DSJC1000.5	(–,83)	1102	89	90	309.7	89	90	962.5	90	91	303.5	90	91	496.9	93	93	546.3	90	91	409.7	97	98	981.1	86	86	514.8
DSJC125.9	(–,30)	12	44	44	0.1	44	44	0.1	44	44	0.1	44	44	0.2	44	44	0.3	44	44	2	44	44	9.9	44	45	3.4
DSJC250.9	(–,72)	60	72	72	3.8	72	72	28.2	72	72	5.6	72	72	26.7	72	73	6.3	72	72	26.6	74	74	39	75	77	12
DSJC500.9	(–,126)	398	127	127	234.4	128	129	180.8	127	128	82.3	128	129	127	129	130	154	127	128	377.2	134	135	340.3	132	132	204.4
DSJC1000.9	(–,224)	2693	226	227	1983.5	230	232	1869.2	227	228	2245	230	230	2382.1	233	234	1621.1	226	229	2401.4	245	247	2143.7	232	233	125.9
flat300_20_0	(20,20)	112	20	20	0.3	20	20	0.3	20	20	0.4	20	20	0.6	20	20	0.6	20	20	1.1	33	34	76.9	20	20	2.9
flat300_26_0	(26,26)	89	26	26	5.5	26	26	16.1	26	26	16.6	26	26	19.8	33	33	4.3	32	33	4.3	35	35	53.5	33	34	2.9
flat300_28_0	(28,31)	61	31	32	3.4	31	31	54.6	31	32	3.3	31	32	7.8	33	33	7.2	33	33	5.1	35	36	17.3	33	34	2.9
flat1000_50_0	(50,50)	1076	85	86	957.4	50	78	1004.9	88	88	729.5	87	88	713.3	50	50	636.3	86	88	470.3	95	96	939.4	84	86	359.3
flat1000_60_0	(60,60)	1119	88	89	245.2	87	88	918.5	89	90	128.2	89	90	372.3	90	91	719.2	88	89	1014.6	96	97	624.5	87	87	235.4
flat1000_76_0	(76,83)	1147	88	89	618.1	88	89	957.6	89	90	188.7	90	90	712	92	92	605	89	90	399	96	97	869	87	87	306
le450_5a	(5,5)	230	5	5	0.1	5	5	0.1	5	5	0.1	5	5	0.9	5	5	0.2	5	7	0	6	7	0	6	7	47.4
le450_5b	(5,5)	232	5	5	0.3	5	5	0.5	5	5	0.6	5	5	0.6	5	5	0.3	6	7	0	6	6	59.6	7	7	37.1
le450_5d	(5,5)	191	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	6	16.4
le450_15a	(15,15)	68	15	15	0.2	15	15	3.4	15	15	0.1	15	15	15	15	15	2.2	16	16	0	16	16	0	16	17	9.3
le450_15b	(15,15)	76	15	15	0.1	15	15	0.2	15	15	0.1	15	15	5.8	15	15	0.3	16	16	0	16	16	0	16	16	18.9
le450_15c	(15,15)	45	16	16	13.5	15	15	19.5	15	15	19.1	15	16	8	15	15	5.9	23	23	0	23	23	0	19	21	216.4
le450_15d	(15,15)	42	16	16	21.7	15	16	13	15	15	20.3	15	16	7.1	15	15	7.8	22	23	0	22	23	0	20	21	189.6
le450_25c	(25,26)	56	26	26	0.7	26	27	0	26	26	2	26	27	0.1	26	26	18	27	28	0	27	28	0	27	28	32.3
le450_25d	(25,26)	59	26	26	0.5	26	27	0	26	26	0.8	26	27	0.2	26	26	4.7	28	28	0	27	28	0	27	27	38.1

TABLE 63.1 *Continued*

	Bench.	time	$TS_{1\text{-ex}}$			HEA			ILS			MC-$TS_{1\text{-ex}}$			GLS			SA_{Kempe}			TS_{VLSN}			XRLF		
Instance	$(\chi,\widehat{\chi}^{best})$	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.
latin_square_10	(–,99)	1242	103	104	617.8	106	107	889.4	103	104	510.4	104	105	458.4	102	103	214.9	101	102	369.9	111	114	798.4	117	118	970.7
qg.order100	(100,100)	12102	100	100	17.9	100	100	18.5	100	100	18.3	100	100	19.9	100	100	36.6	100	101	14.8	100	100	875.1	100	101	3971.9
queen6_6	(7,7)	2	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	1.5	7	7	0
queen7_7	(7,7)	4	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	2.1	7	7	0
queen8_12	(12,–)	7	12	12	0	12	12	0	12	12	0	12	12	0	12	12	0	12	12	0.1	12	12	0.4	12	12	0.9
queen8_8	(9,9)	5	9	9	0	9	9	0	9	9	0	9	9	0	9	9	0	9	9	0.1	9	10	0	9	9	0.2
queen9_9	(10,10)	6	10	10	0	10	10	0	10	10	0	10	10	0	10	10	0	10	10	0.1	10	11	0	10	10	0.4
queen10_10	(11,11)	10	11	11	0.1	11	11	0.1	11	11	0	11	11	0.1	11	11	0.8	11	12	0.1	12	12	0.1	11	11	0.9
queen11_11	(11,12)	14	12	12	0.1	12	12	0.1	12	12	0.2	12	12	0.2	12	13	0	12	13	0.1	13	13	0.4	12	12	2
queen12_12	(12,12)	18	13	13	1	13	13	1.4	13	13	0.9	13	13	3.8	13	14	0	14	14	0.2	14	14	1.1	13	13	13.3
queen13_13	(13,14)	22	14	14	2.9	14	14	2.3	14	14	1.3	14	14	13.2	15	15	0	15	15	0.3	15	15	2.7	14	14	21.4
queen14_14	(14,–)	21	15	16	0	15	16	0	15	15	20	15	16	0	16	16	0	16	16	0.5	16	16	5	15	15	32.2
queen15_15	(15,17)	24	16	17	0	16	17	0	16	16	23.9	16	17	0	17	17	0	17	17	0.8	17	17	5.7	16	17	23.9
queen16_16	(16,18)	24	18	18	0	18	18	0	18	18	0	18	18	0	18	18	0	17	18	1.2	18	18	19.5	17	17	33.4
wap01a	(–,–)	412	43	44	1.2	43	44	1.6	43	44	1.5	42	42	217.1	42	42	55	44	45	107.8	44	46	30.8	47	48	131.6
wap02a	(40,–)	318	42	43	0.7	42	43	0.8	42	42	251.6	41	42	4.9	41	41	159.9	43	43	97.8	43	44	0.5	46	47	150.8
wap03a	(–,–)	1395	46	47	3.8	46	47	4.5	46	46	365.2	45	47	5.8	44	44	782	46	47	198.9	47	48	339.1	50	51	884.3
wap04a	(40,–)	2125	44	44	170	45	45	2.8	44	44	484.3	43	44	31.2	43	43	833.6	45	46	1.6	45	46	1.8	47	49	1073.5
wap06a	(40,–)	138	41	42	0.5	42	42	0.7	42	42	0.5	42	43	0.2	40	41	8.1	42	44	0.2	42	43	4.7	44	44	24.3
wap07a	(–,–)	341	43	44	0.7	43	43	1.9	43	44	0.7	42	43	30.9	42	42	215.2	44	45	9.1	44	45	39	47	47	80.7
wap08a	(40,–)	373	42	43	10.3	42	43	1.4	43	43	56.1	42	44	0.7	42	42	41.4	45	45	0.4	44	45	0.4	46	47	89.4
abb313GPIA	(9,10)	328	9	9	4.1	9	9	26.4	9	9	0.9	9	9	30.7	9	9	1.1	11	11	0	11	11	0.1	12	13	36.4
ash331GPIA	(4,4)	200	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	5	5	25.8
ash608GPIA	(4,4)	633	4	4	0.1	4	4	0.2	4	4	0.1	4	4	0.2	4	4	0.1	4	5	0	4	4	409.3	5	5	27.4
ash958GPIA	(5,4)	1627	4	4	0.5	4	4	0.5	4	4	0.6	4	4	0.9	4	4	0.4	5	5	0	4	5	0	5	5	121.5
will199GPIA	(7,7)	31	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	8	43.9

Note: Given are the chromatic number and best known colorings $(\chi,\widehat{\chi}^{best})$, the maximal computation time for the algorithms in seconds (time), the best and the median coloring found by an algorithm, and the median computation time for reaching a solution with median (solution) quality. The computational experiments were run on a machine with a 2 GHz AMD Athlon MP 2400+ processor, 256 KB cache and 1 GB of RAM memory.

From these results one can draw the following conclusions. Most importantly, there are strong differences in the relative order of the algorithms among the various instance classes and, hence, it is not possible to declare any single algorithm to be the best performing one. On the uniform random graphs, $\mathsf{TS}_{\mathsf{1\text{-}ex}}$, ILS, and HEA are the most competitive algorithms. HEA is the significantly best performing algorithm on the Flat graphs, while on the Leighton graphs ILS and GLS are the best ones. The largest variation in the relative order of the algorithms appears to be due to GLS; GLS is the best or among the best algorithms for Leighton, Jacobian estimation, and WAP graphs, but on the other classes of graphs its performance is significantly worse than, for example, that of $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. Two further results are interesting. First, the exploration of large neighbourhoods in $\mathsf{TS}_{\mathsf{VLSN}}$ does not pay off; in fact, it is among the worst performing algorithms. Further analysis showed that this is mainly due to the higher computational cost per search step [17]. Second, on most instance classes XRLF performs rather poorly and it is among the best algorithms only on the Queens graphs. This contradicts somehow the reputation it gained, which, however, is mainly due to its very good performance on large uniform random graphs with edge density 0.5. Finally, note that the performance of HEA is worse than that reported in Ref. [40]. We verified that this difference is mainly due to our experimental setup and the usage of a single parameter setting; in Ref. [40], HEA was tuned for each specific graph and even the value of k when solving the k-coloring problem. When the implementation of HEA was fine-tuned, our implementation roughly matched the results presented in Ref. [40]. Across all the instances, the very good performance of $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ is most noteworthy, because it is also one of the algorithms that are among the most easy ones to implement.

63.8 Summary

This chapter gives an overview of the main SLS algorithms described in the literature dedicated to the GCP. Most of these SLS algorithms follow the strategy of keeping the number of colors fixed to a value k and trying to minimise the number of conflicts. The optimisation of the GCP is then tackled by solving a series of k-coloring problems. Among the algorithms that follow this strategy, a simple tabu search algorithm based on the restricted 1-exchange neighbourhood is a very robust and fast approach that achieves competitive results for many instance classes. Other SLS algorithms either show better performance only on a few instance classes (e.g., GLS on the WAP instances or Jacobian estimation graphs) or when very high computation times are available (e.g., HEA when appropriately tuned). On most classes of graphs, SLS algorithms also outperform exact algorithms for the GCP. However, exact algorithms appear to be feasible if the chromatic number is equal or very close to the size of the largest clique of the graph [17].

A promising direction for future research on the GCP appears to be the integration of exact and SLS algorithms, given that they show particular advantages for different instance classes. Another challenge for research on SLS algorithms for the GCP is to get a better understanding of the performance of these algorithms in dependence of instance features. Insights into this relationship may help to increase the robustness of the algorithms across the various instance classes and may finally lead to new developments and possibly even better performing algorithms.

Acknowledgments

This work was supported in part by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106 and by a European Community Marie Curie Fellowship, contract HPMF-CT-2001. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication. Irina Dumitrescu acknowledges support of the Canada Research Chair in distribution management, HEC Montreal and Thomas Stützle support of the Belgian FNRS, of which he is a research associate.

References

[1] Jensen, T. R. and Toft, B., *Graph Coloring Problems*, Wiley, 1994.

[2] Allen, M., Kumaran, G., and Liu, T., A combined algorithm for graph-coloring in register allocation, in *Proc. Computational Symp. on Graph Coloring and its Generalizations*, Ithaca, New York, 2002, p. 100.

[3] Barnier, N. and Brisset, P., Graph coloring for air traffic flow management, in *Proc. 4th Int. Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, Le Croisic, France, 2002, p. 133.

[4] Gamst, A., Some lower bounds for a class of frequency assignment problems, *IEEE Trans. Vehicular Tech.*, Vol. 35(1), 8, 1986.

[5] Zymolka, A., Koster, A. M. C. A., and Wessäly, R., Transparent optical network design with sparse wavelength conversion, in *Proc. 7th IFIP Working Conf. on Optical Network Design & Modelling*, Budapest, Hungary, 2003, p. 61.

[6] de Werra, D., An introduction to timetabling, *Eur. J. Oper. Res.*, 19(2), 151, 1985.

[7] Leighton, F. T., A graph coloring algorithm for large scheduling problems, *J. Res. Nat. Bureau Stand.*, 84(6), 489, 1979.

[8] Karp, R. M., Reducibility among combinatorial problems, in *Complexity of Computer Computations*, Miller, R. E., and Thatcher, J. W., Eds., Plenum Press, New York, 1972, p. 85.

[9] Feige, U. and Kilian, J., Zero knowledge and the chromatic number, *JCSS*, 57(2), 187, 1998.

[10] Halldórsson, M. M., A still better performance guarantee for approximate graph coloring, *Inf. Process. Lett.*, 45(1), 19, 1993.

[11] Paschos, V. T., Polynomial approximation and graph-coloring, *Computing*, 70(1), 41, 2003.

[12] Brélaz, D., New methods to color the vertices of a graph, *CACM*, 22(4), 251, 1979.

[13] Caramia, M. and Dell'Olmo, P., Bounding vertex coloring by truncated *multistage* branch and bound, *Networks*, 44(4), 231, 2004.

[14] Mendez Diaz, I. and Zabala, P., A branch-and-cut algorithm for graph coloring, in *Proc. Computational Symp. on Graph Coloring and its Generalizations*, Ithaca, New York, 2002, p. 55.

[15] Gomes, C., and Shmoys, D., Completing quasigroups or latin squares: A structured graph coloring problem, in *Proc. Computational Symp. on Graph Coloring and its Generalizations*, Ithaca, New York, 2002, p. 22.

[16] Mehrotra, A. and Trick, M. A., A column generation approach for graph coloring, *INFORMS J. Comput.*, 8(4), 344, 1996.

[17] Chiarandini, M., Stochastic Local Search Methods for Highly Constrained Combinatorial Optimization Problems, Ph.D. thesis, Computer Science Department, Darmstadt University of Technology, Germany, 2005.

[18] Johnson, D. S. and Trick, M. A., Eds., *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, Vol. 26 DIMACS Series on Discrete Mathematics and Theoretical Computer Science, AMS, Providence, RI, 1996.

[19] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C., Optimization by simulated annealing: an experimental evaluation: Part II, graph coloring and number partitioning, *Oper. Res.*, 39(3), 378, 1991.

[20] Achlioptas, D. and Naor, A., The two possible values of the chromatic number of a random graph, *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, ACM Press, New York, NY, USA, p. 587.

[21] Johri, A. and Matula, D. W., Probabilistic Bounds and Heuristic Algorithms for Coloring Large Random Graphs, Technical Report 82-CSE-6, Southern Methodist University, Dallas, TX, 1982.

[22] Luczak, T., The chromatic number of random graphs, *Combinatorica*, 11(1), 45, 1991.

[23] Culberson, J., Beacham, A., and Papp, D., Hiding our colors, in *Proc. CP'95 Workshop on Studying and Solving Really Hard Problems*, Cassis, France, 1995, p. 31.

[24] Lewandowski, G. and Condon, A., Experiments with parallel graph coloring heuristics and applications of graph coloring, in *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, Vol. 26, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, AMS, Providence, RI, 1996, p. 309.

[25] Hossain, S. and Steihaug, T., Graph coloring in the estimation of mathematical derivatives, in *Proc. Computational Symp. on Graph Coloring and its Generalizations*, Ithaca, New York, 2002, p. 9.

[26] Knuth, D. E., *The Stanford GraphBase: A Platform for Combinatorial Computing*, ACM press, 1993.

[27] Mizuno, K. and Nishihara, S., Toward ordered generation of exceptionally hard instances for graph 3-colorability, in *Proc. Computational Symp. on Graph Coloring and its Generalizations*, Ithaca, New York, 2002, p. 1.

[28] Cheeseman, P., Kanefsky, B., and Taylor, W. M., Where the really hard problems are, in *Proc. of IJCAI*, Morgan Kaufmann Publishers, San Mateo, CA, USA, 1991, p. 331.

[29] Fleurent, C. and Ferland, J. A., Genetic and hybrid algorithms for graph coloring, *Ann. Oper. Res.*, 63, 437, 1996.

[30] Hertz, A. and de Werra, D., Using tabu search techniques for graph coloring, *Computing*, 39, 345, 1987.

[31] Morgenstern, C. and Shapiro, H., Coloration neighbourhood structures for general graph coloring, *Proc. of SODA*, SIAM, 1990, p. 226.

[32] Costa, D., Hertz, A., and Dubois, O., Embedding of a sequential procedure within an evolutionary algorithm for coloring problems in graphs, *J. Heuristics*, 1(1), 105, 1995.

[33] Morgenstern, C., Distributed coloration neighbourhood search, in *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, Vol. 26, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, AMS, Providence, RI, 1996, p. 335.

[34] Chand, A., A constraint based generic model for representing complete university timetabling data, in *Proc. of PATAT'05*, Pittsburgh, PA, 2004, p. 125.

[35] de Werra, D., Heuristics for graph coloring, *Comput. Suppl.*, 7, 191, 1990.

[36] González-Velarde, J. L. and Laguna, M., Tabu search with simple ejection chains for coloring graphs, *Ann. Oper. Res.*, 117(1–4), 165, 2002.

[37] Avanthay, C., Hertz, A., and Zufferey, N., A variable neighborhood search for graph coloring, *Eur. J. Oper. Res.*, 151(2), 379, 2003.

[38] Minton, S., Johnston, M. D., Philips, A. B., and Laird, P., Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, *Artif. Intelligence*, 58(1–3), 161, 1992.

[39] Dorne, R. and Hao, J. K., A new genetic local search algorithm for graph coloring, in *Proc. of PPSN-V*, Eiben, A. E. et al., Eds., Lecture Notes on Computer Science, 1498, Springer, Berlin, 1998, p. 745.

[40] Galinier, P. and Hao, J. K., Hybrid evolutionary algorithms for graph coloring, *J. Comb. Optimization*, 3(4), 379, 1999.

[41] Stützle, T., *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*, *DISKI*, 220, Infix, Sankt Augustin, Germany, 1999.

[42] Voudouris, C., Guided Local Search for Combinatorial Optimization Problems, Ph.D. thesis, University of Essex, Department of Computer Science, Colchester, UK, 1997.

[43] Lourenço, H. R., Martin, O., and Stützle, T., Iterated local search, in *Handbook of Metaheuristics*, Glover, F. and Kochenberger, G., Eds., Kluwer Academic Publishers, Dordrecht, 2002, p. 321.

[44] Chiarandini, M. and Stützle, T., An application of iterated local search to the graph coloring problem, in *Proc. Computational Symp. on Graph Coloring and its Generalizations*, Ithaca, New York, 2002, p. 112.

[45] Paquete, L. and Stützle, T., An experimental investigation of iterated local search for coloring graphs, in *Applications of Evolutionary Computing*, Cagnoni, S. et al., Eds., Lecture Notes on Computer Science, 2279, Springer, Berlin, 2002, p. 122.

[46] Glass, C. A. and Prügel-Bennett, A., A polynomially searchable exponential neighbourhood for graph colouring, *J. Oper. Res. Soc.*, 56(3), 324, 2005.

[47] Culberson, J. C. and Luo, F., Exploring the k-colorable landscape with iterated greedy, in *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, Vol. 26, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 1996, p. 245.

[48] Davis, L., Order-based genetic algorithms and the graph coloring problem, in *Handbook of Genetic Algorithms*, Davis, L., Ed., Van Nostrand Reinhold, New York, 1991, p. 72.

[49] Marino, A. and Damper, R. I., Breaking the symmetry of the graph colouring problem with genetic algorithms, in *Late Breaking Papers at the GECCO Conf.*, Las Vegas, NV, 2000, p. 240.

[50] Galinier, P., Hertz, A., and Zufferey, N., Adaptive memory algorithms for graph colouring, in *Proc. Computational Symp. on Graph Coloring and its Generalizations*, Ithaca, New York, 2002, p. 75.

[51] Eiben, A. E., Hauw, J. K., and Van Hemert, J. I., Graph coloring with adaptive evolutionary algorithms, *J. Heuristics*, 4(1), 25, 1998.

[52] Hamiez, J.-P. and Hao, J.-K., Scatter search for graph coloring, in *Artificial Evolution*, Collet, P. et al., Eds., Lecture Notes on Computer Science, 2310, Springer, Berlin, 2001, p. 168.

[53] Laguna, M. and Martí, R., A GRASP for coloring sparse graphs, *Comput. Optimization Appl.*, 19(2), 165, 2001.

[54] Costa, D. and Hertz, A., Ants can colour graphs, *J. Oper. Res. Soc.*, 48(3), 295, 1997.

[55] Blöchliger, I. and Zufferey, N., A Reactive Tabu Search Using Partial Solutions for the Graph Coloring Problem, Technical Report 04/03, Ecole Ploytechnique Fédérale de Lausanne, Recherche Opérationelle Sud-Est, Lausanne, Switzerland, 2004.

[56] Chams, M., Hertz, A., and De Werra, D., Some experiments with simulated annealing for coloring graphs, *Eur. J. Oper. Res.*, 32(2), 260, 1987.

[57] Conover, W. J., *Practical Nonparametric Statistics*, 3rd ed. Wiley, New York, 1999.

64

On Solving the Maximum Disjoint Paths Problem with Ant Colony Optimization

Maria J. Blesa
Technical University of Catalonia

Christian Blum
Technical University of Catalonia

64.1 Introduction

The efficient use of modern communication networks depends on our capabilities for solving a number of demanding algorithmic problems, some of which are concerned with the allocation of network resources to individual connections. One of the basic operations in communication networks consists in establishing routes for *connection requests* between physically separated network endpoints that wish to establish a connection for information exchange. Many connection requests occur simultaneously in a network, and it is desirable to establish routes for as many requests as possible. In many situations, either because of technical constraints or just to improve the communication, it is required that no two routes interfere with each other, which implies not to share network resources such as links or switches. This scenario can be modeled as follows. Let $G = (V, E)$ be an edge-weighted undirected graph representing a network in which the nodes represent the hosts and switches, and the edges represent the links. Let $T = \{(s_j, t_j) \mid j = 1, \dots, |T|;\ s_j \neq t_j \in V\}$ be a list (of size ITI) of *commodities*, that is, pairs of nodes in G, representing endpoints demanding to be connected by a path in G. T is said to be *realizable* in G if there exist mutually edge-disjoint (respectively vertex-disjoint) paths (EDP) from s_j to t_j in G, for every $j = 1, \dots,$ ITI.

The question whether T is realizable was early known to be NP-complete [1] in arbitrary graphs. The problem remains NP-complete for specific types of graphs such as planar graphs [2,3], series-parallel graphs (a.k.a. partial 2-trees) [4], and grid graphs [5,6]. For several types of graphs, this problem belongs to the class of APX-hard problems [7–10]. This fact explains the notorious hardness of the EDP problem in terms of approximation, despite the attention and effort that researchers have put on it. Interestingly, for the specific case of complete graphs, we are not aware of any inapproximability results. In particular, it is not even known whether the problem in complete graphs is APX-hard.

The combinatorial optimization version of this problem consists in satisfying as many of the requests as possible, which is equivalent to finding a realizable subset of T of maximum cardinality. A solution S to the combinatorial optimization problem is a set of disjoint paths, in which each path satisfies the connection request for a different commodity. For any solution S, the objective function value $f(S)$ is defined as

$$f(S) = |S| \tag{64.1}$$

In general, the "disjointness" of paths may refer to nodes or edges. We decided to consider the latter case because it seems of higher importance in practical applications. We henceforth refer to our problem as the maximum EDP problem. The EDP problem is obtained from the more general *unsplittable flow problem* by considering the demands, profits, and capacities to be one. In the extreme case, in which the list of commodities is composed by repetitions of the same pair (s, t), the problem is known as *edge-disjoint Menger problem* [11].

The EDP problem is interesting for different research fields such as combinatorial optimization, algorithmic graph theory, and operations research. It has a multitude of applications in areas such as real-time communications, VLSI-design, scheduling, bin packing, load balancing, and it has recently been brought into focus in works discussing applications to routing and admission control in modern networks, namely large-scale, high-speed, and optical networks [12–15]. Concerning real-time communications, the EDP problem is very much related to survivability and information dissemination. Concerning survivability, having several disjoint paths available may avoid the negative effects of possible failures occurring in the base network. Furthermore, to communicate via multiple disjoint paths can increase the effective bandwidth between pairs of nodes, reduce congestion in the network, and increase the velocity and the probability of receiving the information [16,17]. This becomes especially important nowadays. Due to the type of information that circulates over networks (e.g., media files), which requires fast, qualified, and reliable connections.

In general, there is a lack of efficient algorithms for tackling the EDP problem. Only some greedy approaches (which we will mention in Section 64.3) and a preliminary ant colony optimization (ACO) approach [18] exist for tackling the problem. The greedy approaches are used as approximation algorithms for theoretical purposes, but the quality of the solutions they obtain are susceptible to improvement. The direct application of a basic ACO scheme to a problem achieves sometimes quite good results. However, the performance of such an algorithm can often be improved by applying some additional features to the search process, especially when a rather unusual problem such as the EDP is tackled. Based on the (basic) approach [18], we have evolved a more sophisticated ACO algorithm (see Ref. [19] for details). We present and evaluate this ACO approach in the remainder of this chapter.

64.2 A Greedy Approach

A greedy heuristic is a constructive algorithm that builds a solution step by step starting from an empty solution. At each construction step, an element from a finite set of solution components is added to the current partial solution. The element to be added is chosen at each step according to some greedy function, which lends the name to the algorithm. A characteristic feature of the greedy algorithms is that, once a decision is made on which element to add, this decision is never reconsidered again. Advantages of greedy heuristics are that they are usually easy to implement and that they are fast in execution. In contrast, the disadvantage is that the quality of the solutions provided by greedy algorithms is often far from being optimal.

Greedy algorithms are often used as *approximation algorithms* to solve optimization problems with a *guaranteed performance.* With this aim, some greedy algorithms were proposed for the EDP problem; examples are the *simple greedy algorithm* [20], its constrained variant the *bounded-length greedy algorithm* [20–24], and the *greedy path algorithm* [25,26]. Due to its lower time complexity when compared to the other greedy approaches we decided to implement the simple greedy algorithm (henceforth denoted by SGA) and a multistart version, which we both outline in the following.

The simple greedy algorithm (SGA)—see Algorithm 64.1 is a natural way of approximating the EDP problem that works as follows. It starts with an empty solution S. Then, it proceeds through the commodities in the order that is given as input. For routing each commodity $T_j \in T$, it considers the graph G

without the edges that are already in the paths of the solution S under construction. The shortest path (with respect to the number of edges) between s_j and t_j is assigned as path for the commodity $T_j = (s_j, t_j)$. In addition to its simplicity, the SGA algorithm can be naturally considered an online algorithm and then, the lower bounds of Ref. [27] would imply that it does not achieve a good performance ratio on graphs such as trees and two-dimensional meshes. However, it works well for many other types of graphs.

Algorithm 64.1 Simple greedy algorithm (SGA) for the EDP problem

```
INPUT: a problem instance $(G, T)$, consisting of a graph $G$ and a commodity list $T$
$S \leftarrow \emptyset$, $\hat{E} \leftarrow E$
for $j = 1, \ldots, |T|$ do
  if $s_j$ and $t_j$ can be connected by a path in $G = (V, \hat{E})$ then
    $P_j \leftarrow$ shortest path from $s_j$ to $t_j$ in $G = (V, \hat{E})$
    $S \leftarrow S \cup P_j$, $\hat{E} \leftarrow \hat{E} \setminus \{e \mid e \in P_j\}$
  end if
end for
OUTPUT: the solution $S$
```

Observe that the SGA algorithm is deterministic and that the quality of the solutions it provides depends heavily on the order in which the commodities are treated. A simple way of overcoming that dependence on the order is to develop a multistart version of the SGA by permuting—for each restart—the order of the commodities. This approach is pseudocoded in Algorithm 64.2, in which N_{perm} the number of restarts, S_i the solution under construction in the embedded SGA, and S_{best} the best solution found so far. In the following, we refer to this algorithm as *multistart greedy algorithm* (MSGA).

Algorithm 64.2 Multistart simple greedy algorithm (MSGA) for the EDP problem

```
INPUT: a problem instance $(G, T, N_{perm})$, where $N_{perm}$ is the number of restarts
$S_{best} \leftarrow \emptyset$, $T_{(1)} \leftarrow T$                    {$T_{(i)}$ denotes the $i$th permutation of $T$}
for $i = 1$ to $N_{perm}$ do
  $S_i \leftarrow$ Simple Greedy Algorithm SGA$(G, T_{(i)})$              {See Algorithm 64.1}
  if $f(S_i) > f(S_{best})$ then $S_{best} \leftarrow S_i$ end if
  if $i < N_{perm}$ then
    $\pi \leftarrow$ random permutation of size $|T|$
    $T_{(i+1)} \leftarrow (\pi(1), \pi(2), \ldots, \pi(|T| - 1), \pi(|T|))$
  end if
end for
OUTPUT: $S_{best}$
```

64.2.1 An Example where SGA and the Multistart Greedy Algorithm Fail

As we have commented, the main disadvantage of some existing greedy algorithms that approximate hard problems is the low quality of the solutions they provide. Due to the deterministic greedy decisions that they take during the solution construction, it is sometimes not possible for them to find an existing optimal solution. This is also the case for SGA and MSGA presented here.

Consider, for example, the instance of the EDP problem depicted in Figure 64.1, which consists of the depicted graph and the set $T = \{(v_1, v_7), (v_8, v_{14}), (v_{15}, v_{21})\}$ of three commodities to join. The optimal

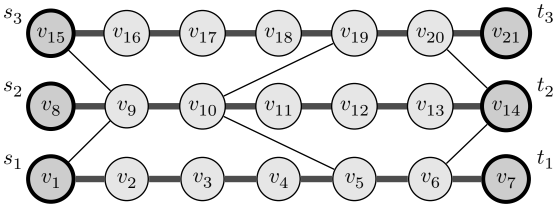


FIGURE 64.1 Example of an instance of the EDP problem (with $T = \{(v_1, v_7), (v_8, v_{14}), (v_{15}, v_{21})\}$) for which neither the SGA nor the MSGA greedy algorithm can find the solution of size 3 emphasized with bold font.

solution in which all three commodities are connected is also shown in bold font in the Figure 64.1.[1] Observe, however, that there is no way for any of the greedy algorithms, neither SGA nor MSGA, to find the solution of size greater than two. Since these greedies are based on the shortest paths (in terms of the number of edges), the algorithms will tend to connect the commodities through nonconsecutively numbered vertices. For example, when trying to connect first the commodity (v_1, v_7), the SGA algorithm will establish the path $\{v_1, v_9, v_{10}, v_5, v_6, v_7\}$. This excludes edge $\{v_9, v_{10}\}$ as a possibility for being used in other paths, which makes it impossible to build disjoint paths simultaneously for the remaining two commodities, independently of which one is built next. Analogous situations occur when starting from any of the other two commodities. Thus, since no possible permutation of the commodities would provide a solution of size three, neither the SGA nor the MSGA will find the optimal solution of size three.

64.3 An Ant Colony Optimization Approach

Ant colony optimization [28,29] is a metaheuristic for solving hard combinatorial optimization problems. Apart from the application to static combinatorial optimization problems (see Ref. [30] for an extensive overview), the method has also gained recognition for the applications to adaptive routing in static and dynamic communication networks [31,32]. Ant Colony Optimization algorithms are composed by independently operating computational units, namely *artificial ants*, that generate a global perspective without the necessity of direct interaction. This exclusive use of local information is an advantageous and desirable feature when applications in large-scale environments are concerned in which the computation of global information is often too costly. This property makes ACO algorithms a natural choice for the application to the EDP problem.

Ant colony optimization is inspired by the foraging behavior of real ants. While walking from food sources to the nest and vice versa, ants deposit a chemical substance called *pheromone* on the ground. When they decide about a direction to go, they choose probabilistically paths marked by strong pheromone concentrations. This behavior is the basis for a cooperative interaction, which leads to the emergence of shortest paths between food sources and their nest. In ACO algorithms, artificial ants incrementally construct a solution by adding appropriately defined solution components to the current partial solution. Each of the construction steps is a probabilistic decision based on local information, which is represented by the *pheromone* information.

In the following, we outline our ACO approach, which is based on a decomposition of the EDP problem. Each problem instance $\mathcal{P} = (G, T)$ of the EDP problem can be naturally decomposed into $|T|$ subproblems $\mathcal{P}_j = (G, T_j)$, with $j \in \{1, \ldots, |T|\}$, by regarding the task of finding a path for a commodity $T_j \in T$ as a problem itself. With respect to this problem decomposition, we use a number of $|T|$ ants each of which is assigned to exactly one of the subproblems. Therefore, the construction of a solution consists of each ant

[1] This solution is found by our ACO algorithm, which is presented next, in a small amount of time (less than 30 ms).

building a path P_j between the two endpoints of her commodity T_j. Obviously, the subproblems are not independent as the set of $|T|$ paths constructed by the ants should be mutually edge-disjoint.

64.3.1 Ant Solutions and Pheromone Model

Our algorithm will deal with solutions that contain a path for each commodity. A solution S constructed by the $|T|$ ants is a set of nonnecessarily edge-disjoint paths. We henceforth refer to them as *ant solutions*, in contrast to the EDP solutions, which only consist of disjoint paths. From each ant solution, a valid EDP solution can be produced by iteratively removing the path, which has most edges in common with the remaining paths, until all remaining paths are mutually edge-disjoint.

The objective function $f(\cdot)$ of the problem (see Eq. [64.1]) is characterized by having many plateaus when it is applied to ant solutions. This is because many ant solutions have the same number of disjoint paths. Thus, a consequence of decomposing the EDP problem is the need to define a more fine-grained objective function $f^a(\cdot)$ for ant solutions. Therefore, referring to $f(S)$ as a *first criterion*, we introduce a *second criterion* $C(S)$, which is defined as follows:

$$C(S) = \sum_{e \in E} \left(\mathsf{max} \left\{ 0, \left(\sum_{P_j \in S} \delta^j(S, e) \right) - 1 \right\} \right) \quad \text{where } \delta^j(S, e) = \begin{cases} 1, & e \in P_j \in S \\ 0, & \text{otherwise} \end{cases}$$

This second criterion quantifies the degree of nondisjointness of an ant solution. If all the paths in a solution S are edge-disjoint, $C(S)$ is zero. In general, $C(S)$ increases when increasing the number of edges in S which are common to more than one path. Therefore, based on the idea that *the fewer edges are shared in a solution, the closer the solution is to disjointness*, a function $f^a(\cdot)$ that differentiates between ant solutions can be defined as follows. For two ant solutions S and S', it holds that

$$f^a(S) > f^a(S') \Leftrightarrow \underbrace{(f(S) > f(S'))}_{\text{1st criterion}} \text{ or } \underbrace{((f(S) = f(S') \text{ and } (C(S) < C(S'))}_{\text{2nd criterion}} \tag{64.2}$$

The problem decomposition as described above requires that we use a pheromone model τ^j for each subproblem $\mathcal{P}_j$. Each pheromone model τ^j consists of a pheromone value, that is, a positive numerical value, τ_e^j for each edge $e \in E$. The set of $|T|$ pheromone models is henceforth denoted by $\tau = \{\tau^1, \ldots, \tau^{|T|}\}$. The pheromone values are bounded between 0 and 1, since our ACO algorithm is implemented in the hypercube framework [33]. Furthermore, to prevent the algorithm from converging to a solution, we borrow an idea from the so-called $\mathcal{MAX}$-$\mathcal{MIN}$ Ant Systems [34] and forbid the extreme pheromone values of 0 or 1 by introducing new pheromone value limits $\tau_{\min} = 0.001$ and $\tau_{\max} = 0.999$.

64.3.2 Algorithmic Framework

In the following, we give a high-level description of an ACO algorithm for the EDP problem (see Algorithm 64.3). The main procedures used by the algorithm are explained in detail in the following section. First, all the variables are initialized. In particular, the pheromone values are set to their initial value $\tau_{\min}$ by the procedure InitializePheromoneValues(τ), which initializes all the pheromone values $\tau_e^j \in \tau^j \in \tau$ to the value $\tau_{\min}$. Second, N_{sols} ant solutions are constructed per iteration. To construct a solution, each ant applies the function ConstructSolution(G,π) (see Section 64.3.2.1 for details), where π is a permutation of T. At each iteration, the first of those N_{sols} ant solutions is constructed with the identity permutation, that is, by sending the ants in the order in which the commodities are given in T. However, for each further ant solution construction in the same iteration, π is randomly generated by the function GenerateRandomPermutation($|T|$) to avoid bias.

Algorithm 64.3 ACO algorithm for the EDP problem

```
INPUT: a problem instance $(G, T)$
$S_{gbest} \leftarrow \emptyset$, $S_{pbest} \leftarrow \emptyset$, $c_{crit1} \leftarrow 0$, $c_{crit2} \leftarrow 0$, all_update $\leftarrow$ FALSE
InitializePheromoneValues($\tau$)
while termination conditions not met do
    $\pi \leftarrow (1, 2, \ldots, |T| - 1, |T|)$
    for $i = 1$ to $N_{sols}$ do
        $S_i \leftarrow$ ConstructSolution($G$,$\pi$)                                {See Algorithm 64.4}
        if $i < N_{sols}$ then $\pi \leftarrow$ GenerateRandomPermutation($|T|$) end if
    end for
    Choose $S_{ibest} \in \{S_i \mid i = 1, \ldots, N_{sols}\}$ s.t. $f^a(S_{ibest}) \geq f^a(S)$, $\forall S \in \{S_i \mid i = 1, \ldots, N_{sols}\}$
    if $f(S_{ibest}) > f(S_{gbest})$ then $S_{gbest} \leftarrow S_{ibest}$ end if
    if $f^a(S_{ibest}) > f^a(S_{pbest})$ then
        $c_{crit1} \leftarrow c_{crit1} + 1$, $c_{crit2} \leftarrow 0$, $S_{psave} \leftarrow S_{pbest}$, $S_{pbest} \leftarrow S_{ibest}$
        if $f(S_{ibest}) > f(S_{psave})$ then
            $S_{update} \leftarrow$ ExtractDisjointPaths($S_{pbest}$)                {Update for first phase}
            $c_{crit1} \leftarrow 0$, all_update $\leftarrow$ FALSE
        end if
        if all_update then $S_{update} \leftarrow S_{pbest}$ end if                {Update for second phase}
    else $c_{crit2} \leftarrow c_{crit2} + 1$
    end if
    if all_update and $c_{crit2} > c2_{\max}$ then
        $S_{pbest} \leftarrow$ DestroyPartially($S_{pbest}$)                        {Escape mechanism}
        $S_{update} \leftarrow$ ExtractDisjointPaths($S_{pbest}$)
        $c_{crit2} \leftarrow 0$, $c_{crit1} \leftarrow 0$
    else if not all_update then all_update $\leftarrow (c_{crit1} > c1_{\max})$
    end if
    UpdatePheromoneValues($\tau$, $S_{update}$)
end while
OUTPUT: the EDP solution generated from the best solution $S_{gbest}$
```

Three different ant solutions are kept in the algorithm: S_{ibest} is the *iteration-best* solution, that is, the best ant solution generated in the current iteration, and S_{gbest} is the *best-so-far* solution, that is, the best ant solution found since the start of the algorithm. In addition to them, an ant solution S_{pbest} is also kept, which is the *currently best* solution, that is, the best ant solution generated since the last escape action (see Section 64.3.2.3). The values of these three variables are always kept updated. Additionally there is the S_{update} solution, which is generated from S_{pbest} and which is used for updating the pheromone values.

The search process has two differentiated phases (see Section 64.3.2.2) and two variables, c_{crit1} and c_{crit2}, are introduced to control them. The variable c_{crit1} determines the first phase by counting the number of successive iterations without improvement of the first criterion of the objective function. The variable c_{crit2} counts the number of successive iterations without improvement of the second criterion, thus defining the second phase. Limits $c1_{\max}$ (for c_{crit1}) and $c2_{\max}$ (for c_{crit2}) are used to determine when the algorithm should change phases.[2] The direct repercussion of the phase distinction is the selection of edges whose pheromone is updated, that is, the construction of S_{update} from S_{pbest}. When the algorithm is in the first

[2] After parameter tuning we chose a setting of $c1_{\max} = c2_{\max} = 20$.

phase only the disjoint paths of solution S_{pbest} are used for updating, but when the algorithm is in the second phase all paths of S_{pbest} are used for updating. Additionally, the escape mechanism might be applied by destroying S_{pbest} partially (see Section 64.3.2.3).

Finally, the pheromone values are updated in the method UpdatePheromoneValues(τ, S_{update}) depending on the edges of the paths included in S_{update}. The algorithm is iterated until some opportunely defined termination conditions are satisfied, and it returns the EDP solution generated from the ant solution S_{gbest}.

In the Sections 64.3.2.1–64.3.2.3, we explain in more detail the features concerning the solution construction, the search procedure and its different search phases, and the escape mechanism of our algorithm, respectively.

64.3.2.1 Solution Construction

The solution construction is performed in method ConstructSolution(G,π), whose high-level description is shown in Algorithm 64.4. That construction is done as follows: at each construction step, each ant moves from the node where it is currently located to a neighboring node by traversing one of the available edges that is not already in its path $P_{\pi(j)}$ under construction, and is not labeled forbidden by a backtracking move. Note that with this strategy the ant will find a path between its source and its destination, if there exists one. Otherwise, the ant returns an empty path. This way of constructing the solution emulates that the ants build concurrently their paths, in contrast to a sequential way in which, for each commodity, a path between its endpoints would be built completely before the next commodity is considered.

Algorithm 64.4 Method ConstructSolution(G,π) of Algorithm 64.3.

```
INPUT: a graph $G$ from a problem instance $(G, T)$, and a permutation $\pi$ of $T$.
$S \leftarrow \emptyset$, $nb\_paths\_finished \leftarrow 0$, $j \leftarrow 0$
for $i = 1$ to $|T|$ do $P_{\pi(i)} \leftarrow \emptyset$ end for
repeat
  if not isFinishedPath($P_{\pi(j+1)}$) then
    $P_{\pi(j+1)} \leftarrow$ ExtendOneStepPath($P_{\pi(j+1)}$,$\tau^{\pi(j+1)}$)
    if isFinishedPath($P_{\pi(j+1)}$) then
      $nb\_paths\_finished \leftarrow nb\_paths\_finished + 1$
      $S \leftarrow S \cup \{P_{\pi(j+1)}\}$
    end if
  end if
  $j \leftarrow (j + 1) \bmod |T|$
until ($nb\_paths\_finished = |T|$)
EvaporatePheromone($\tau$,$S$)
OUTPUT: an ant solution $S$
```

The procedures of Algorithm 64.4 are detailed in the following:

- is Finished Path (P_i).[3] This method returns a Boolean value indicating whether the path P_i is finished, that is, whether a path could be established from s_i to t_i.
- ExtendOneStepPath(P_i,τ^i).[3] The path P_i passed as parameter is the path under construction by the ith ant. For constructing a path between the endpoints of the commodity (s_i, t_i), an ant first chooses randomly to start either from the source s_i or from the target t_i; this is done when the path

[3] For readability, we substitute $\pi(j + 1)$ in the description of the functions by i.

P_i is empty. Afterward this method either tries to extend the path P_i by adding exactly one edge or, it performs a backtracking step. Backtracking is done in case the ant finds itself in a node in which all the incident edges have been used, or if all the incident edges are labeled forbidden.

Once one of the two endpoints of the commodities is chosen as starting point, the remaining endpoint becomes the so-called *goal node* and will be denoted by v_g. Additionally, let us denote by v_c the current node, and by $\mathcal{I}^*_{v_c}$ the set of allowed edges in G, that is, those incident to v_c that are not used yet in the path and not labeled as forbidden. The length of the shortest path between two vertices u and v in G is henceforth denoted by $\sigma(u, v)$ and it is measured in terms of the number of edges.

From the set $\mathcal{I}^*_{v_c}$ of allowed edges, only the two best edges will be actually considered as candidates. This is called a *candidate list strategy* in ACO. The best two edges are those that maximize the value of the following expression:

$$\tau_e^j \mathbf{p}(D_e)\mathbf{p}(U_e)$$

where $\mathbf{p}(D_e)$ is a value that determines the influence of the distance from v_c via u to the goal vertex v_g, and $\mathbf{p}(U_e)$ a value that determines the influence of the overall usage of edge e, which is the information whether e is already used in the path of another ant for the same solution. The terms $\mathbf{p}(D_e)$ and $\mathbf{p}(U_e)$ are defined as follows:

$$\mathbf{p}(D_{e=\{v_c,u\}}) \leftarrow \frac{(\sigma(u, v_g) + w(e))^{-1}}{\sum\limits_{e'=\{v_c,u'\}\in\mathcal{I}^\star_{v_c}} (\sigma(u', v_g) + w(e'))^{-1}}$$

$$\mathbf{p}(U_e) \leftarrow \frac{U(e)^{-1}}{\sum\limits_{e'\in\mathcal{I}^*_{v_c}} U(e')^{-1}} \quad \text{in which} \quad U(e) = \begin{cases} 2, & e \text{ already used in } S_i \\ 1, & \text{otherwise} \end{cases}$$

Thus, using this candidate list strategy, we can reduce the set of allowed edges in $\mathcal{I}^*_{v_c}$ and just consider a new two-cardinality set $\mathbb{I}^*_{v_c} = \{e_1^*, e_2^*\}$, where e_1^* is the best edge in $\mathcal{I}^*_{v_c}$, that is,

$$e_1^* = \{v_c, u\} \leftarrow \mathsf{argmax}\,\{\tau_e^j \mathbf{p}(D_e)\mathbf{p}(U_e) \mid e \in \mathcal{I}^*_{v_c}\}$$

and e_2^* the second best edge in $\mathcal{I}^*_{v_c}$.

At each construction step, the choice of where to move to has a certain probability p to be done deterministically, and a certain probability $1 - p$ to be chosen probabilistically among the elements in $\mathbb{I}^*_{v_c}$. This is a feature that we adopt from a particularly effective ACO variant called Ant Colony System (ACS [35]). In 75% of the cases, the next edge to join the path $P_{\pi(k)}$ under construction will be e_1^*, while in the remaining 25% of the cases, the next edge is chosen from $\mathbb{I}^*_{v_c}$ according to the following transition probabilities:

$$\mathbf{p}(e \mid \mathbb{I}^*_{v_c}) = \frac{\tau_e^j \mathbf{p}(D_e)\mathbf{p}(U_e)}{\sum\limits_{e'\in\mathbb{I}^*_{v_c}} \tau_{e'}^j \mathbf{p}(D_{e'})\mathbf{p}(U_{e'})}, \quad \forall e \in \mathbb{I}^*_{v_c} \tag{64.3}$$

In general, if the probability of doing a deterministic construction step is too high, there is the danger that the algorithm gets stuck in low-quality regions of the search space. However, doing deterministic construction steps bears the potential of leading the algorithm quite quickly to good areas of the search space. Concerning the composition of the transition probabilities, the use of the pheromone information τ_e^j ensures the flexibility of the algorithm, whereas the use of $\mathbf{p}(D_e)$ ensures a bias toward short paths, and $\mathbf{p}(U_e)$ ensures a bias toward disjointness of the $|T|$ paths constituting a solution.

- EvaporatePheromone(τ, S). After every ant has constructed its path and the solution S is completed, we apply another feature of ACS, namely the evaporation of some amount of pheromone from the edges that were used by the ants. Given a solution S, the evaporation is done as follows:

$$\tau_e^j \leftarrow \begin{cases} (1-\varepsilon)\cdot\tau_e^j, & e \in P_{\pi(j)} \in S, \quad j = 1, \ldots, |T| \\ \tau_e^j, & \text{otherwise} \end{cases} \tag{64.4}$$

The reason for this pheromone evaporation is the desire to diversify the search in each iteration.[4]

64.3.2.2 Search with Distinguished Phases

In general, the pheromone update procedure is an important component of every ACO algorithm. In fact, it determines to a large degree the failure or the success of the algorithm. Most of the existing generic variants of ACO only differ in the pheromone update. In the case of the EDP application, we propose a pheromone updating scheme that is based on the idea that, to maintain a higher degree of freedom for finding also edge-disjoint paths for the commodities that initially prove to be problematic, it might be better not to use the nondisjoint paths for updating the pheromone at the beginning of the search. Therefore, we propose a two-phases search process based on the two criteria of function $f^a(\cdot)$ (see Eq. [64.2]): A first phase of the algorithm in which the algorithm will try to improve the first criterion of $f^a(\cdot)$ (while disregarding the second one) and only disjoint paths are used for updating the pheromone values; the first phase is followed by a second phase which is initiated when no improvements of the first criterion can be found over a certain time bounded by $c1_{\max}$. In this second phase, the algorithm will try to improve the second criterion of $f^a(\cdot)$ and all the paths are used for updating the pheromone values. Once the second phase leads to an improvement also in terms of the first criterion, the algorithm changes back to the first phase.

In the first phase, the solution S_{update} that is used for updating the pheromone values is obtained by applying function ExtractDisjointPaths(S_{pbest}), which implements the process of returning a valid EDP solution from the ant solution S_{pbest} as explained in Section 64.3.1. In the second phase, the solution S_{update} that is used for updating the pheromone values is a copy of the current solution S_{pbest}, including possibly nondisjoint paths. If for a number of $c2_{\max}$ iterations the second criterion could not be improved neither, then some of the paths from the EDP solution that can be produced from S_{pbest} are deleted from S_{pbest}. This action can be seen as a mechanism to escape from the current area of the search space and it is explained in Section 64.3.2.3.

After the solution S_{update} is constructed, the pheromone of the edges conforming its paths are updated as follows:

$$\tau_e^j \leftarrow \max\left\{\tau_e^j + \rho\left(1-\tau_e^j\right),\ \tau_{\max}\right\} \quad \forall\, e \in P_j \in S_{update} \tag{64.5}$$

where $\rho \in (0, 1)$ is a constant value which is called *learning rate* in algorithms that is implemented in the hypercube framework.[5] This pheromone update is performed in function UpdatePheromoneValues (τ,S_{update}).

64.3.2.3 Escape Mechanism

One of the main problems of metaheuristic search procedures is to detect situations in which the search process gets stuck, that is, when some local minimum is reached. Most of the successful applications

[4] After parameter tuning we chose a setting of $\varepsilon = 0.10$.

[5] For all our experiments we have set ρ to 0.1.

incorporate algorithm features to escape from these situation once they are detected. In case of our algorithm for the EDP problem, we propose as escape mechanism the partial destruction of the disjoint part of the solution, which is used for updating the pheromone values. This escape mechanism is implemented through the function DestroyPartially(S_{pbest}), whose pseudocode is outlined in Algorithm 64.5. This mechanism is triggered once the algorithm is unable to improve the currently best solution for a number of subsequent applications of first and second phases, since that situation indicates that the search process is stuck in a localized area. Similar ideas are applied in backtracking procedures, or in the perturbation mechanism of local search-based methods, such as iterated local search or variable neighborhood search [36].

Algorithm 64.5 Method DestroyPartially(S_{pbest}) of Algorithm 64.3. ExtractDisjointPaths(S_{pbest}) implements the process of returning a valid EDP solution from an ant solution as explained in Section 64.3.1. The method Cost(S_{temp}) returns the number of disjoint paths in S_{temp}. The method ChooseLongestPath(S_{temp}) return the longest disjoint path of S_{temp}. The method ResetPheromoneModel(τ^i) resets to $\tau_{\min}$ all the pheromone values of the pheromone model τ^i, i.e., $\tau_e^i \leftarrow \tau_{\min},\ \forall e \in E$.

```
INPUT: an ant solution $S_{pbest}$
$S_{pbest} \leftarrow$ ExtractDisjointPaths($S_{pbest}$)
$nb_{paths} \leftarrow \left\lceil \frac{1}{4} \cdot \text{Cost}(S_{pbest}) \right\rceil$
while ($nb_{paths} > 0$) do
    $P_i \leftarrow$ ChooseLongestPath($S_{pbest}$)
    $S_{pbest} \leftarrow S_{pbest} \setminus \{P_i\}$
    $nb_{paths} \leftarrow nb_{paths} - 1$
    ResetPheromoneModel($\tau^i$)
end while
OUTPUT: the solution $S_{pbest}$ partially destroyed
```

One-fourth of the disjoint paths composing solution S_{pbest} are destroyed. The disjoint paths to be destroyed are chosen according to their lengths, giving priority to the longest paths, that is, those paths with the highest number of edges. The idea behind this choice is that, the longer a path is, the more restrictions it introduces to assure disjointness of the paths that still conflict with others. Thus, by removing the longest disjoint paths, the number of total edges available is maximized.[6]

64.4 Experiments

We present the experimental evaluation of our ACO approach in comparison to the results obtained by the greedy approaches that we outlined in Section 64.2. As commented before, the ACO algorithm presented here resulted from a detailed algorithm design process that started with a very simple ACO approach [18], which was improved and enriched until the algorithm explained in the present work was obtained. In the same process, the values for those parameters involved in the algorithm were fixed (after a careful tuning)

[6] Other options were tried, both considering different values for the percentage of solution's paths to be destroyed, as well as a random selection of the paths to be destroyed. However, these options were not providing us with better results (see Ref. [19]).

TABLE 64.1 Parameters Used for the Generation of Network Topologies with BRITE [37]

Graph	$\|V\|$	Model	Node Placement	m
bl-wr2-wht2.10-50	(10,50)	(Waxman, Waxman)	(random, heavy-tailed)	(2, 2)
AS-BA.R-Wax.v100e190	(20, 5)	(Barabási–Albert [39], Waxman)	(random, random)	(2, 2)
AS-BA.R-Wax.v100e217	(10,10)	(Barabási–Albert [39], Waxman)	(random, random)	(2, 2)

Notes: In each value tuple (X_{as}, X_{rou}), X_{as} is the value of the parameter at the AS level, and X_{rou} is the value of the parameter at the router level. Parameter m specifies the number of links for each new node that is added while constructing the topology. For all the graphs, the growth type (i.e., how nodes join the topology) is incremental. In graph bl-wr2-wht2.10-50, the edge connections between the AS level and the router level are introduced using the Waxman probability model [38] with parameters $\alpha = 0.15$ and $\beta = 0.20$; in graphs AS-BA.R-Wax.v100e190 and AS-BA.R-Wax.v100e217 both levels are interconnected by choosing edges at random.

to the values provided here. The experiments to be presented in the following were done with those settings (see Ref. [19] for more detail)

All the algorithms were implemented in C++ and compiled using GCC 2.95.2 with the -o3 option. The experiments have been run on a PC with Intel(R) Pentium(R) 4 processor at 3.06 GHz and 900 Mb of memory running a Linux operating system. Moreover, our algorithms were all implemented on the same data structures. Information about the shortest paths in the respective graphs is provided to all of them as input. Notice however that, while the greedy approaches need to partially recompute this information after the routing of each commodity, this is not necessary for our ACO algorithm.

64.4.1 Problem Instances

In the following, we present the set of benchmark instances that we used to experimentally evaluate our ACO approach. This set of instances includes graphs representing different communication network topologies. Recall that an instance of the EDP problem consists of a graph and a set of commodities.

Concerning the graphs, we adopt graph3 and graph4 from Ref. [18], whose structure resembles parts of the communication network of the Deutsche Telekom AG, Germany. Additionally, we include graphs that we created with the network topology generator BRITE [37] according to the parameter values specified in Table 64.1. These three generated graphs are named bl-wr2-wht2.10-50, AS-BA.R-Wax.v100e190, and AS-BA.R-Wax.v100e217. They consist of a two-level top-down hierarchical topology (autonomous system level plus router level), which are typical for Internet topologies. Table 64.2 summarizes the main features and quantitative measures of all the considered graphs.

For each of the five graphs we have randomly generated different sets of commodities. Hereby, we made the size of the commodity sets dependent on the number of vertices of the graph. For each graph $G = (V, E)$ we generated 20 different instances with $0.10|V|$, $0.25|V|$, and $0.40|V|$ commodities. This makes a sum of 60 instances for each graph and 300 instances altogether.

TABLE 64.2 Main Quantitative Measures of Our Benchmark Graphs

				Degree			Clustering
Graph	$\|V\|$	$\|E\|$	Minimum	Average	Maximum	Diameter	Coefficient
graph3 [18]	164	370	1	4.51	13	16	0.226161
graph4 [18]	434	981	1	4.52	20	22	0.155547
bl-wr2-wht2.10-50 [18]	500	1020	2	4.08	13	23	0.102385
AS-BA.R-Wax.v100e190	100	190	2	3.80	7	11	0.378524
AS-BA.R-Wax.v100e217	100	217	2	4.34	8	13	0.411119

64.4.2 Experiments and Results

We applied the algorithms presented in this chapter (namely SGA, MSGA, and ACO) to all 300 instances exactly once. First, we applied MSGA with 50 restarts (i.e., $N_{perm} = 50$) to each of the 300 instances. The computation time of MSGA was used as a maximum CPU time limit for the ACO algorithm. We present the results as averages over the 20 instances of each combination of graph and commodity number in Table 64.3. The layout of this table is explained in its caption.

Concerning the comparison between SGA and MSGA, we observe a clear advantage of MSGA. This means that the order in which the commodities are treated is crucial in achieving a good performance. However, as there is no obvious way of determining a good commodity order beforehand, the only way of exploiting this knowledge is by randomly permuting the commodity list and running MSGA. The price we have to pay for exploiting this knowledge is the increased computation time.

When comparing SGA and MSGA with the ACO, we can observe that in 11 out of 15 cases the ACO approach beats the greedy approaches. The ACO approach is on average 4.69% better than MSGA, and in one case (graph4, 173 commodities) it is even 15.07% better. Additionally, the ACO approach needs in general less computation time than the greedy approaches. This advantage in computation time increases with increasing number of commodities. Exceptions are some of the results for small numbers of commodities, namely for 10% of the number of nodes. For this combination MSGA has often slight advantages over the ACO approach. Therefore, we recommend to use a greedy approach when easy problem instances are concerned, but to use the ACO approach for instances with a higher number of commodities, since then a clear advantage of the latter is observed in comparison to MSGA both in quality and time.

An additional analysis concerns the run-time behavior of the algorithms. Figure 64.2 shows that the ACO approach finds relatively good solutions already after a very short computation time. In general, already the first solutions produced by the ACO are quite good, whereas the greedy approaches reach a comparable solution quality only much later in time. This property of our ACO approach is a desirable feature in the context of communication networks since the quality of the solutions that are found after a short execution time might be often sufficient in practice. Also of interest is Figure 64.3 where a representative example of the usefulness of ACO's escape mechanism is shown. Also the evolution of the second criterion as a measure for disjointness can be clearly observed.

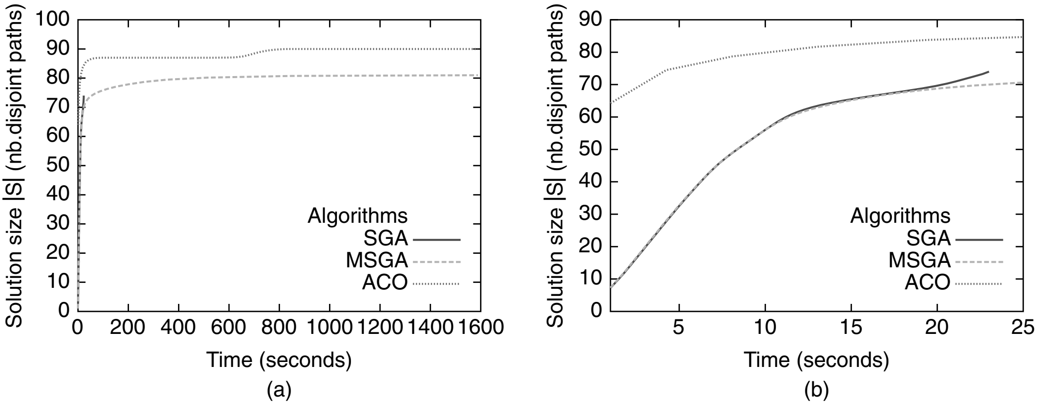


FIGURE 64.2 A representative example of the run-time behavior of the algorithms presented in this work. All the curves are smoothed with gnuplots' *sbezier* function. (a) Example of the evolution in time of the quality of the solution S_{gbest}. The behavior shown here corresponds to the application to one of the 20 instances composed by graph4 and a list of 173 commodities. (b) Zoom on the first 25 of part (a) of this figure. The time needed to obtain good solutions is clearly smaller for the ACO approach. Note that in the first seconds the performance of the SGA and the MSGA are identical due to dealing with the same permutation of the commodities.

TABLE 64.3 Comparison of the Results Obtained by the SGA, the MSGA, and the ACO Algorithm.

Graph	Number of Commodities	SGA			MSGA			ACO			
		$\overline{q}$	σ	$\overline{t}$	$\overline{q}$	σ	$\overline{t}$	$\overline{q}$	σ	$\overline{t}$	average CPU time
graph3	16	15.30	0.781	0.566	15.70	0.557	0.960	**15.70**	0.557	0.457	30.582
graph3	41	29.00	2.864	1.298	**32.00**	2.302	25.235	31.80	1.990	27.953	79.619
graph3	65	33.70	2.777	2.156	37.60	2.577	49.267	**40.30**	2.571	57.899	126.945
graph4	43	40.50	1.628	12.121	**42.05**	1.024	95.744	41.45	1.284	168.871	237.520
graph4	108	58.10	4.194	31.138	64.10	3.064	697.456	**68.15**	2.725	730.436	1656.475
graph4	173	66.75	4.846	49.281	73.95	3.542	974.350	**85.10**	3.534	1111.982	2603.872
bl-wr2-wht2.10-50	50	19.70	2.238	17.926	22.55	2.397	318.518	**24.10**	1.947	155.899	971.488
bl-wr2-wht2.10-50	125	34.15	4.464	46.387	38.10	4.369	1004.462	**42.30**	4.540	344.092	2425.090
bl-wr2-wht2.10-50	200	46.70	4.961	62.158	50.85	4.892	1151.197	**56.30**	5.245	847.415	3124.550
AS-BA.R-Wax.v100e190	10	8.75	0.942	0.114	**9.10**	0.943	0.579	8.95	0.973	0.611	6.665
AS-BA.R-Wax.v100e190	25	12.30	1.900	0.280	14.25	1.374	4.809	**14.85**	1.195	3.718	16.740
AS-BA.R-Wax.v100e190	40	15.45	2.500	0.443	17.95	1.624	7.796	**19.45**	1.936	4.121	26.850
AS-BA.R-Wax.v100e217	10	7.00	1.225	0.103	**8.05**	0.921	0.427	7.88	0.927	0.164	6.892
AS-BA.R-Wax.v100e217	25	11.40	1.882	0.300	13.60	1.463	4.330	**13.83**	1.579	1.816	17.622
AS-BA.R-Wax.v100e217	40	14.60	1.685	0.497	17.00	1.949	9.833	**17.80**	1.646	2.212	28.318

Notes: The first column gives the name of the graph and the second column the number of the commodities, which are obtained as the 10, 25, and 40% of the number of nodes of the graphs. For each algorithm there are three columns reporting on the average results obtained for the 20 instances of each combination of graph topology and number of commodities. The first of these three columns (headed by $\overline{q}$) shows the average of the values of the best solutions found for the 20 instances. Such an average is in boldface when the result is the best in the comparison. In case of ties the computation time decides. The second column provides the standard deviation of the 20 values used to compute $\overline{q}$, and the third column (headed by $\overline{t}$ reports on the average time (in seconds) needed to find the best solution values for the 20 instances. Finally, the last column shows the average computation time of MSGA.

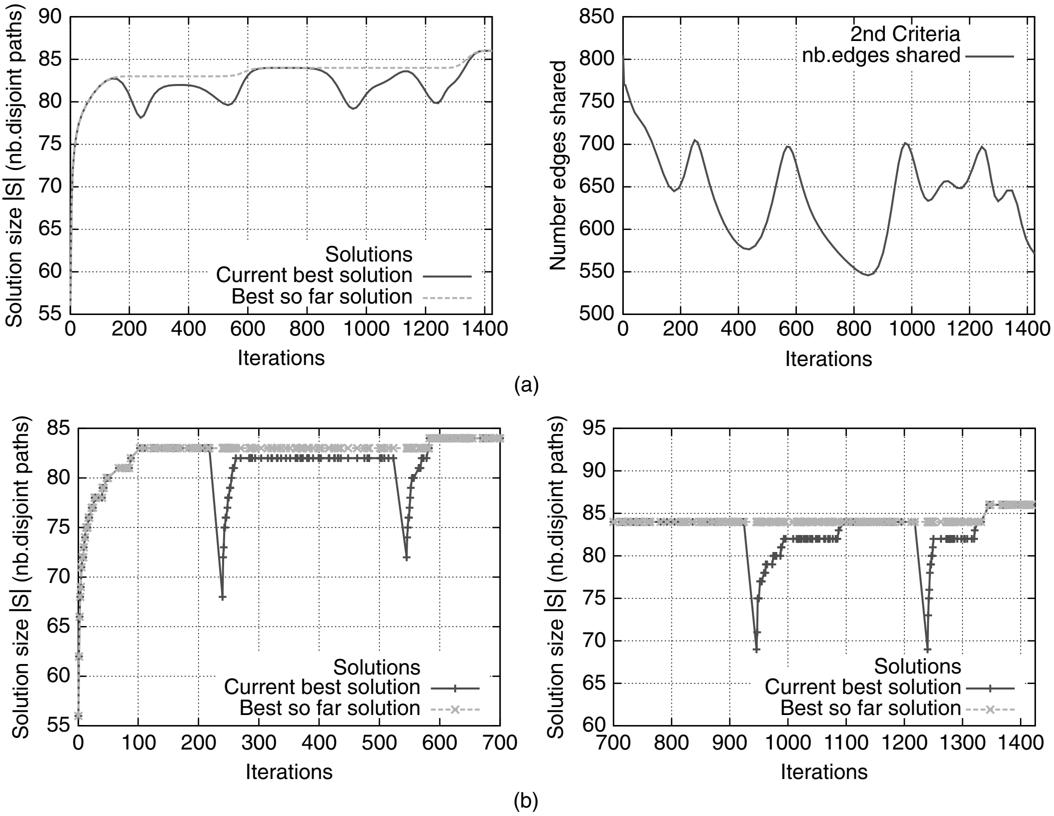


FIGURE 64.3 A representative example of the behavior of the ACO algorithm. The effect of the mechanism for the partial destruction of the current best solution can be clearly observed. It is also interesting to observe the evolution of the second criterion as a measure for disjointness. (a) Example of the evolution of the quality of the current best solution S_{pbest} and the best-so-far solution S_{gbest} during the search (*left*), and the number of shared edges (2nd criterion) of the solution S_{pbest} (*right*). The behavior shown here corresponds to the application to one of the 20 instances composed by graph4 and a list of 173 commodities. All the curves are smoothed with gnuplots' *sbezier* function. (b) Zoom on the 700 first (*left*) and the 700 last (*right*) iterations of part (a) of this figure. On the *left*, the best solution found is quickly improved. At about iteration 250, the algorithm destroys part of the S_{pbest} solution, which produces an instantaneous worsening in the quality (*left*); another solution destruction takes places around iteration 550, which helps in achieving an improvement soon afterward (*left*). Analogous effects can be observed around iteration 950 and 1250 (*right*). In part (a) (*right*) of this figure, we can observe that there exists an (inverted) relation between the number of edges shared and the quality of the solutions obtained. Thus validating our choice of the 2nd criterion as a part of the objective function.

Acknowledgments

This work is partially supported by EU Programmes under contract numbers IST-2004-15964 (AEOLUS), COST-295 (DYNAMO), and by the Spanish CICYT project TIC-2002-04498-C05-03 (TRACER). Additionally, as a "*Ramón y Cajal*" research fellow. Christian Blum acknowledges support from the Spanish Ministry of Science and Technology.

The authors would like to thank Martin Oellrich from the Combinatorial Optimization & Graph Algorithms group of the Technische Universität Berlin for providing them with graph3 and graph4.

References

[1] Karp, R., Reducibility among combinatorial problems, in *Complexity of Computer Computations*, Miller, R. E. and Thatcher, J. W., Eds., Plenum Press, New York, 1972, p. 85.

[2] Middendorf, M. and Pfeiffer, F., On the complexity of the disjoint path problem, *Combinatorica*, 13(1), 97, 1993.

[3] Vygen, J., NP-completeness of some edge-disjoint paths problems, *Disc. Appl. Math.*, 61(1), 83, 1995.

[4] Nishizeki, T., Vygen, J., and Zhou, X., The edge-disjoint paths problem is np-complete for series-parallel graphs, *Disc. Appl. Math.*, 115(1–3), 177, 2001.

[5] Kramer, M. and van Leeuwen, J., The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits, in *Advances in Computing Research*, Preparata, F. P., Ed., *Vol. 2: VLSI Theory*, JAI Press, London, UK, 1984, p. 129.

[6] Marx, D., Eulerian disjoint paths problem in grid graphs is NP-complete, *Disc. Appl. Math.*, 143(1–3), 336, 2004.

[7] Garg, N., Vazirani, V., and Yannakakis, M., Primal-dual approximation algorithms for integral flow and multicut in trees, *Algorithmica*, 18(1), 3, 1997.

[8] Ma, B. and Wang, L., On the inapproximability of disjoint paths and minimum steiner forest with bandwidth constraints, *JCSS*, 60(1), 1, 2000.

[9] Erlebach, T., Approximation algorithms and complexity results for path problems in trees of rings, in *Int. Symp. on Math. Foundations of Comp. Sci.*, Lecture Notes in Computer Science, Vol. 2136, Springer, Berlin, 2001, p. 351.

[10] Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., and Yannakakis, M., Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems, *JCSS*, 67(3), 473, 2003.

[11] Menger, K., Zur allgemeinen Kurventheorie, *Fundamenta Mathematicae*, 10, 96, 1927.

[12] Awerbuch, R., Gawlick, R., Leighton, F. T., and Rabani, Y., On-line admission control and circuit routing for high performance computing and communication, *Proc. FOCS,* 1994, p. 412.

[13] Raghavan, P. and Upfal, E., Efficient all-optical routing, *Proc. STOC,* 1994, p. 134.

[14] Aggarwal, A., Bar-Noy, A., Coppersmith, D., Ramaswami, R., Schieber, B., and Sudan, M., Efficient routing and scheduling algorithms for optical networks, *Proc. SODA,* 1994, p. 412.

[15] Aumann, Y. and Rabani, Y., Improved bounds for all-optical routing, *Proc. SODA,* 1995, p. 567.

[16] Hromkovič, J., Klasing, R., Stöhr, E., and Wagener, H., Gossiping in vertex-disjoing paths mode in d-dimensional grids and planar graphs, *Eur. Symp. on Algorithms,* Lecture Notes in Computer Science, Vol. 726, Springer, Berlin, 1993, p. 200.

[17] Sidhu, D., Nair, R., and Abdallah, S., Finding disjoint paths in networks, *ACM SIGCOMM Comp. Comm. Rev.*, 21(4), 43, 1991.

[18] Blesa, M. and Blum, C., Ant colony optimization for the maximum edge-disjoint paths problem, *Proc. Eur. Workshop on Evolutionary Computation in Communications, Networks, and Connected Sys.*, Lecture Notes in Computer Science, Vol. 3005, Springer, Berlin, 2004, p. 160.

[19] Blesa, M. and Blum, C., Finding edge-disjoint paths with artificial ant colonies, TR LSI-05-13-R, ALBCOM research group, Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 2005. `www.lsi.upc.edu/dept/techreps/techreps.html`.

[20] Kleinberg, J., Approximation Algorithms for Disjoint Paths Problems, Ph.D. thesis, MIT, Cambridge, MA, 1996.

[21] Kolman, P. and Scheideler, C., Simple on-line algorithms for the maximum disjoint paths problem, *Proc. SPAA,* 2001, p. 38.

[22] Kolman, P. and Scheideler, C., Improved bounds for the unsplittable flow problem, *Proc. SODA,* 2002, p. 184.

[23] Carmi, P., Erlebach, T., and Okamoto, Y., Greedy edge-disjoint paths in complete graphs, Goos, G., Hartmanis, J., and van Leeuwen, J., Eds., in *Graph-Theoretic Concepts in Computer Science,* Lecture Notes in Computer Science, Vol. 2880, Springer, Berlin, 2003, p. 143.

[24] Kolman, P., A note on the greedy algorithm for the unsplittable flow problem, *Inform. Proc. Lett.*, 88(3), 101, 2003.

[25] Kolliopoulos, S. and Stein, C., Approximating disjoint-path problems using packing integer programs, *Math. Prog.*, 99(1), 63, 2004.

[26] Chekuri, C. and Khanna, S., Edge disjoint paths revisited, *Proc. SODA,* 2003, p. 628.

[27] Awerbuch, B., Azar, Y., and Plotkin, S., Throughput-competitive online routing, *Proc. FOCS,* 1993, p. 32.

[28] Dorigo, M., *Ottimizzazione, Apprendimento Automatico, ed Algoritmi basati su Metafora Naturale*, Ph.D. thesis, DEI, Politecnico di Milano, Milan, Italy, 1992.

[29] Dorigo, M., Maniezzo, V., and Colorni, A., Ant System: optimization by a colony of cooperating agents, *IEEE Trans. on Syst., Man, and Cybern. B*, 26(1), 29, 1996.

[30] Dorigo, M. and Stützle, T., *Ant Colony Optimization*, MIT Press, Cambridge, MA, 2004.

[31] Di Caro, G. and Dorigo, M., AntNet: distributed stigmergetic control for communications networks, *J. Artif. Intelligence Res.*, 9, 317, 1998.

[32] Di Caro, G., Ducatelle, F., and Gambardella, L. M., AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks, *Eur. Trans. Telecom.*, 16(2), 2005.

[33] Blum, C. and Dorigo, M., The hyper-cube framework for ant colony optimization, *Trans. on Syst., Man, and Cybern.–B*, 34(2), 1161, 2004.

[34] Stützle, T. and Hoos, H., $\mathcal{MAX}$-$\mathcal{MIN}$ ant system, *Future Generation Comp. Sys.*, 16(8), 889, 2000.

[35] Dorigo, M. and Gambardella, L., Ant Colony System: a cooperative learning approach to the traveling salesman problem, *IEEE Trans. Evol. Comput.*, 1(1), 53, 1997.

[36] Blum, C. and Roli, A., Metaheuristics in combinatorial optimization: overview and conceptual comparison, *ACM Comput. Surv.*, 35(3), 268, 2003.

[37] Medina, A., Lakhina, A., Matta, I., and Byers, J., BRITE: Boston University Representative Internet Topoloy Generator, `http://cs-pub.bu.edu/brite/index.htm`, 2001.

[38] Waxman, B., Routing of multipoint connections, *J. Sel. Areas Comm.*, 6(9), 1671, 1988.

[39] Barabási, A. and Albert, R., Emergence of scaling in random networks, *Science*, 509, 1999.

VI

Large-Scale and Emerging Applications

65

Cost-Efficient Multicast Routing in Ad Hoc and Sensor Networks

Pedro M. Ruiz
University of Murcia

Ivan Stojmenovic
University of Ottawa

65.1 Introduction

A mobile ad hoc network (MANET) consists of a number of devices equipped with wireless interfaces. Ad hoc nodes are free to move, and communicate with each other using their wireless interfaces. Communications among nodes which are not within the same radio range are carried on via multihop routing. That is, some of the intermediate nodes between the source and the destination act as relays to deliver the messages. Hence, these networks can be deployed without any infrastructure, making them specially interesting for dynamic scenarios like battlefield, rescue operations, and even as flexible extensions of mobile networks for operators.

Wireless sensor networks (WSNs) follow a similar communication paradigm based on multihop paths. Although wireless sensor nodes are not usually mobile, their limited resources in terms of battery life and computational power pose additional challenges to the routing task. For instance, wireless sensor nodes operate following a duty cycle, allowing them to save energy while they are sleeping. The different timings for sleep and awake periods across sensors make the topology change. In addition, these networks are usually densely populated compared to ad hoc networks, requiring very efficient and scalable mechanisms to provide the routing functions. Examples of such techniques gaining momentum nowadays are geographic routing and localized algorithms in general, in which nodes take individual decisions solely based on the local information about itself and its neighbors. These networks have a lot of potential applications, which is one of the reasons why they are receiving so much attention within the research community.

The most commonly used model for these networks is called "Unit Disk Graph." The network is modeled as an undirected graph $G = (V, E)$, where V is the set of vertices and E the set of edges. The model assumes that the network is two-dimensional (every node $v \in V$ is embedded in the plane) and wireless nodes are represented by vertices of the graph. Each node $v \in V$ has a transmission range r. Let $dist(v_1, v_2)$ be the distance between two vertices $v_1, v_2 \in V$. An edge between two nodes $v_1, v_2 \in V$ exists iff $dist(v_1, v_2) \leq r$ (i.e., v_1 and v_2 are able to communicate directly).

Unicast routing both for MANETs and WSNs can be defined as the process of finding a path in the network to deliver a message from the originator to the destination. As we mentioned before, in these networks such paths are formed by a set of nodes acting as relays. The multicast routing task is similar to the unicast routing except that there are a number of destinations instead of a single node. These destinations are often referred as "receivers" in the literature. In this particular case, the set of relay nodes usually forms a tree, commonly known as "multicast tree." Below, we define more precisely the problem of unicast and multicast routing in these networks.

Definition 65.1

Given a graph $G = (V, E)$, a source node $s \in V$ and a destination node $D \in V$, the unicast routing problem can be defined as finding a set of relay nodes $F \subset V$ s.t. $\{s\} \cup F \cup \{D\}$ is connected.

Similarly, the multicast routing problem can be defined as follows:

Definition 65.2

Given a graph $G = (V, E)$, a source node $s \in V$ and a set of destinations $R \subseteq V$, the multicast routing problem, can be defined as finding a set of relay nodes $F \subset V$ s.t. $\{s\} \cup F \cup R$ is connected.

Of course, routing algorithms are designed to avoid cycles, and usually select paths according to some metric or combination of metrics such as hop count, delays, etc. In fact, most of the existing routing protocols use the hop count as the path selection metric.

The problem of unicast routing is well known and there are many distributed algorithms such as Dijkstra, Bellman-Ford, and so on. For the problem of multicast routing, there are also algorithms to build shortest path trees (SPTs), shared trees, etc. In fact, the problem of the efficient distribution of traffic from a set of senders to a group of receivers in a datagram network was already studied by Deering [1] in the late 1980s. Several multicast routing protocols such as DVMRP [2], MOSPF [3], CBT [4], and PIM [5] have been proposed for IP multicast routing in fixed networks. These protocols have not been usually considered in MANETs because they do not properly support mobility. In the case of mesh networks, one may believe that they can be a proper solution. However, they were not designed to operate on wireless links, and they lead to suboptimal routing solutions which are not able to take advantage of the broadcast nature of the wireless medium (i.e., sending a single message to forward a multicast message to all the next hops rather than replicating the message for each neighbor). Moreover, their routing metrics do not aim at minimizing the cost of the multicast tree, which limits the overall capacity of the mesh network.

Within the next sections, we describe existing multicast routing protocols for ad hoc and sensor networks, and we analyze the issue of computing minimum-cost multicast trees. In fact, we will show the NP-completeness of the problem.

Given that the use of approximation algorithms is fully justified in the multicast routing case, we focus the rest of the chapter on the multicast routing problem, and its approximation algorithms for MANETs and WSNs. The remainder of the chapter is organized as follows: Section 65.2 describes existing multicast routing protocols for MANETs and their inability to approximate minimum-cost multicast trees. Section 65.3 focus on the issue of computing minimum bandwidth multicast trees in wireless ad hoc networks, shows the NP-completeness of the problem, and offers approximation algorithms which offer better performance than Steiner trees. We focus on the problem of geographic multicast routing in Section 65.4. Finally, we provide some discussion and conclusions in Section 65.5.

65.2 Multicast Routing in Ad Hoc Networks

A plethora of protocols have been proposed for multicast routing in MANETs. We focus our discussion on the most representative protocols. They can be classified into tree- or mesh-based depending upon the underlying forwarding structure that they use. Tree-based schemes [6–10] construct a multicast tree from each of the sources to all the receivers using generally an SPT or a shared tree. Mesh-based approaches [11,12] compute several paths among senders and destinations. Thus, when the mobility rate increases they are able to tolerate link breaks better than tree-based protocols at the expense of a usually higher overhead. Hybrid approaches [13,14] try to combine the robustness of mesh-based ad hoc routing and the low overhead of tree-based protocols. Finally, there are stateless multicast protocols [15] in which there is no need to maintain a forwarding state on the nodes (for instance, if the nodes to traverse are included in the data packets themselves). We will not discuss further about the latter category given the very limited applicability of those variants.

Regarding tree-based protocols, AMRIS [6] builds a shared multicast tree among a set of sources and receivers. There is a root node, which is the one with the smallest ID (Sid). These ID numbers are assigned dynamically within a multicast session, and based on these IDs the multicast tree is built. The numbering process starts at the Sid, and other nodes always select an ID being higher than the one of their upstream nodes in the tree. MAODV [7] is an extension of the well-known AODV protocol. The route creation is similar to the RREQ/RREP process in AODV, except that the source unicasts an MACT (Multicast Activation) message through the selected paths, which usually form a SPT based on the hop count.

Regarding mesh-based multicast routing protocols for ad hoc networks, ODMRP [11] works reactively to build a multicast mesh connecting senders and receivers. All the intermediate nodes taking part in the multicast mesh are said to belong to the forwarding group (FG). When a multicast node has data to send and it does not have a route for that multicast group, it starts a periodic broadcasting of JOIN_QUERY (JQ) packets. These messages are propagated through the entire ad hoc network avoiding duplicates, so that every ad hoc node can learn which of his neighbors is in its shortest path to that source. Upon reception of a nonduplicate JQ message, a receiver broadcasts a JOIN_REPLY (JR) message in which it includes the ID of the neighbor selected as the next hop to reach each of the multicast sources. When a node receives a JR message it checks out if its own ID is listed as the selected next hop for any of the sources. If that is the case, then it realizes that it is in the SPT to any of the sources, and it adds itself to the FG by activating its FG_FLAG. In addition, the selected node sends out a JR which he fills with the IDs of its selected neighbors to reach those sources for which it was selected in the received JR message. In this way, the FG is populated until the JR messages reach the source. This whole process is repeated periodically to update the FG over time. Once the mesh is built, data forwarding is very simple. Only those nodes whose FG_FLAG is active are allowed to forward data packets generated by the sources. In addition, in case of receiving the same data packet several times, a node within the FG shall only forward it the first time it is received. CAMP [12] was designed as an extension of the "Core-Based Trees" (CBT; Ref. [4]) protocol. However, unlike CBT in which there was no link redundancy, CAMP builds a multicast mesh to offer a much better performance and resilience in case of link breaks. Whereas in CBT core nodes were used for data forwarding, they are used in CAMP to reduce the overhead for a node to find out a node belonging to the multicast mesh. Data packets are not required to go through core nodes. Thus, given that there can be several core nodes for a multicast group, the tolerance of mobility is increased. CAMP uses a receiver-initiated approach to build the multicast mesh. Using the cores as well-known mesh nodes, CAMP avoids relying on periodic flooding of the network to find multicast routes. However, this comes at the cost of depending upon the unicast routing protocol as well as the need of a mapping service from multicast groups to core nodes. CAMP ensures that the mesh contains all the reverse shortest paths between the source and the recipients. It uses a so-called heartbeat mechanism by which each node periodically monitors its packet cache. If the node finds out that some data packets which it is receiving are not coming from its shortest path neighbor, then it sends a HEARTBEAT message to its successor in the reverse shortest path to the source. The HEARTBEAT triggers a push-join (PJ) message which (if the successor is not a mesh member) forces the successor and all the nodes in the path to join the mesh.

All these solutions are not aimed at minimizing the cost of multicast trees due to the difficulty of computing such trees. In fact, when the goal is to find multicast trees with minimum edge cost, the problem becomes NP-complete and requires heuristic solutions. Such minimum-cost multicast tree is well known as the Steiner tree problem. However, Ruiz and Gomez-Skarmeta [16] showed that ODMRP (and accordingly other similar protocols) can benefit from the use of Steiner trees rather than SPTs and shared trees. Karp [17] demonstrated that the Steiner tree problem is NP-complete even when every link has the same cost using a transformation from the exact cover by 3-sets. There are some heuristic algorithms [18] to compute minimal Steiner trees. For instance, the MST heuristic [19,20] provides a 2-approximation, and Zelikovsky [21] proposed an algorithm which obtains a 11/6-approximation. Recently, Rajagopalan and Vazirani [22] proposed a 3/2-approximation algorithm. Given the complexity of computing this kind of trees in a distributed way, most of the existing multicast routing protocols use SPTs or suboptimal shared trees, which can be easily computed in polynomial time.

Recently Ruiz and Gomez-Skarmeta [23] showed that the Steiner tree is not the best solution for wireless ad hoc networks, and provided some approximation algorithms which will be analyzed in this chapter. The problem of minimizing the bandwidth consumption of a multicast tree in an ad hoc network needs to be reformulated in terms of minimizing the number of data transmissions. By assigning a cost to each link of the graph computing the tree that minimizes the sum of the cost of its edges, existing formulations have implicitly assumed that a given node v needs k transmissions to send a multicast data packet to k of its neighbors. However, in a broadcast medium, the transmission of a multicast data packet from a given node v to any number of its neighbors can be done with a single data transmission. Thus, in ad hoc networks the minimum cost tree is the one which connects sources and receivers by issuing a minimum number of transmissions, rather than having a minimal edge cost. We will discuss further this problem along next sections.

65.3 Minimum Bandwidth Consumption Multicast Tree

When nodes are mobile, such as in traditional MANETs, it is not really interesting to approximate optimal multicast trees. The reason is that by the time the tree is computed, it may no longer exist. However, in some new static ad hoc network scenarios being deployed (i.e., wireless mesh networks) these algorithms may become of utmost relevance. In wireless mesh networks, devices are powered. So, the main concern, rather than being the power consumption, is the proper utilization of the bandwidth. Thus, the computation of bandwidth-optimal multicast trees becomes really important.

Given a multicast source s and a set of receivers R in a network represented by a undirected graph, we are interested in finding the multicast tree with the minimal cost in terms of the total amount of bandwidth required to deliver a packet from s to every receiver.

In wired networks, the computation of such minimum bandwidth consumption multicast tree is equivalent to the Steiner tree over a graph $G = (V, E)$ so that $w(e_i) = b_s, \forall i = 1..|E|$, b_s being the rate at which the source s is transmitting, and $w(e_i)$ the cost of the link number i. Unitary costs (i.e., $w(e_i) = 1, \forall i = 1..|E|$) are usually assumed for simplicity. The bandwidth consumption of such a Steiner tree $T^* = (V^*, E^*)$ is then proportional to $|E^*|$. This means that the problem is NP-complete in wired networks. As we mentioned before, in wireless multihop networks a node can send a message to all neighbors with a single transmission. Thus, the bandwidth consumption is different from the edge cost, and the problem requires being reformulated in terms of the number of transmissions rather than the number of traversed links. To account for the excessive bandwidth consumption due to suboptimal trees, we will define a new metric called “data overhead.”

65.3.1 Problem Formulation

Before going into details, we need some definitions which are used in the sequel.

Definition 65.3

Given a graph $G = (V, E)$, a source $s \in V$, and a set of receivers $R \subset V$, we define the set T as the set of the possible multicast trees in G which connect the source s to every receiver $r_i \in R$. We denote by F_t, the set of relay nodes in the tree $t \in T$, consisting of every nonleaf node, which relays the message sent out by the multicast source. We define a function $C_t : T \rightarrow \mathbb{Z}^+$ so that given a tree $t \in T$, $C_t(t)$ is the number of transmissions required to deliver a message from the source to every receiver induced by that tree.

Lemma 65.1

Given a tree $t \in T$ as defined above, $C_t(t) = 1 + |F_t|$.

Proof
By definition relay nodes forward the message sent out by s only once. In addition, leaf nodes do not forward the message. Thus, the total number of transmissions is one from the source and one from each relay node, making a total of $1 + |F_t|$. □

So, as we can see from Lemma 65.1, to minimize $C_t(t)$ we must somehow reduce the number of forwarding nodes $|F_t|$. Note that some receivers may serve also as relay nodes.

Definition 65.4

Under the conditions of definition 65.3, let $t^* \in T$ be the multicast tree such that $C_t(t^*) \leq C_t(t)$ for any possible $t \in T$. We define the data overhead of a tree $t \in T$, as $\omega_d(t) = C_t(t) - C_t(t^*)$. Obviously, with this definition $\omega_d(t^*) = 0$.

On the basis of the previous definitions, the problem can be formulated as follows. Given a graph $G = (V, E)$, a source node $s \in V$, a set of receivers $R \subset V$, and given $V' \subseteq V$ defined as $V' = R \cup \{s\}$, find a tree $T^* = (V^*, E^*) \subset G$ such that the following conditions are satisfied:

(1) $V^* \supseteq V'$
(2) $C_t(T^*)$ is minimized.

From the condition of T^* being a tree it is obvious that it is connected, which combined with condition (1) establishes that T^* is a multicast tree. Condition (2) is equivalent to $\omega_d(T^*) = 0$ and establishes the optimality of the tree. As we show below, this problem is NP-complete.

65.3.2 NP-Completeness

Ruiz and Gomez-Skarmeta [23] demonstrated that this problem is NP-complete. We show below a similar demonstration based on the inclusion of the Minimim Common Dominating Set (MCDS) as a particular case of the problem.

Theorem 65.1

Given a graph $G = (V, E)$, a multicast source $s \in V$ and a set of receivers R, the problem of finding a tree $T^* \supseteq R \cup \{s\}$ so that $C_t(T^*)$ is minimum is NP-complete.

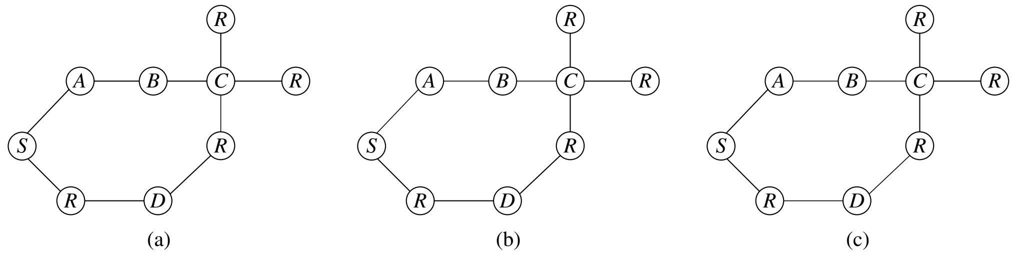
Proof
According to Lemma 65.1, minimizing $C_t(T^*)$ is equivalent to minimizing the number of relay nodes $F \subseteq T^*$. So, the problem is finding the smallest set of forwarding nodes F that connects s to every $r \in R$. If we consider the particular case in which $R = V - \{s\}$, the goal is finding the smallest connected $F \subseteq T^*$ that covers the rest of nodes in the graph $(V - \{s\})$. This problem is one of finding a minimum connected dominating set, which is known to be NP-complete [24]. □

Ruiz and Gomez-Skarmeta [23] also proved the suboptimality of Steiner trees for this particular problem in ad hoc networks. We give a simple example to prove it.

Theorem 65.2

Let $G = (V, E)$ be an undirected graph. Let $s \in V$ be a multicast source and $R \subseteq V$ be the set of receivers. The Steiner multicast tree $T^* \subseteq G$ so that $C_e(T^*)$ is minimal may not be the minimal data-overhead multicast tree.

Proof
It is immediate from the examples shown in Figure 65.1 that the Steiner tree may not minimize the bandwidth consumption, leading to suboptimal solutions in MANETs and WSNs. □



(a) (b) (c)

FIGURE 65.1 Differences in cost for several multicast trees over the same ad hoc network in (a) source path tree with eight edges and six transmissions; (b) Steiner tree with six edges and five transmissions; and (c) minimum data overhead tree with seven edges and four transmissions.

In general, Steiner tree heuristics try to reduce the cost by minimizing the number of Steiner nodes $|\mathbb{S}^*|$. In the next section we will present our heuristics being able to reduce the bandwidth consumption of multicast trees, by just making receivers be leaf nodes in a cost-effective way.

65.3.3 Heuristic Approximations

Given the NP-completeness of the problem, within the next subsections we describe two heuristic algorithms proposed by Ruiz and Gomez-Skarmeta [23] to approximate minimal data-overhead multicast trees. As we learned from the demonstration of Theorem 65.2, the best approach to reduce the data overhead is reducing the number of forwarding nodes, while increasing the number of leaf nodes. The two heuristics presented below try to achieve that trade-off.

65.3.3.1 Greedy-Based Heuristic Algorithm

The first proposed algorithm is suited for centralized wireless mesh networks, in which the topology can be known by a single node, which computes the multicast tree.

Inspired by the results from Theorem 65.2, this algorithm first systematically builds different cost-effective subtrees. The cost-effectiveness refers to the fact that a node v is selected to be a forwarding node only if it covers two or more nodes. That is, if it has two or more multicast receivers as neighbors.

The algorithm shown in Algorithm 65.1, starts by initializing the nodes to cover (aux) to all the sources except those already covered by the source s. Initially the set of forwarding nodes (MF) is empty. After the initialization, the algorithm repeats the process of building a cost-effective tree, starting with the node v which covers the largest number of nodes in "aux." Then, v is inserted into the set of forwarding nodes (MF) and it becomes a node to cover. In addition, the receivers covered by v (Cov(v)) are removed from

Algorithm 65.1 Greedy minimal data overhead algorithm MNT

```
MF ← $\emptyset / * \mathit{mcast - forwarders} * /$
V ← V - {$s$}
aux ← R-Cov(s) + {$s$} $/ * \mathit{nodes - to - cover} * /$
**repeat**
    node ← $\mathit{argmax}_{v \in V}(|\mathrm{Cov}(v)|)$ s.t. Cov($v$)$\geq$2
    aux ← aux-Cov($v$)+{$v$}
    V ← V-{$v$}
    MF ← MF + {$v$}
**until** aux = $\emptyset$ or node = *null*
**if** V!=$\emptyset$ **then**
    Build Steiner tree among nodes in aux
**end if**
```

the list of nodes to cover denoted by “aux.” This process is repeated until all the nodes are covered, or it is not possible to find more cost-effective subtrees. In the latter case, the different subtrees are connected by a Steiner tree among their roots, which are in the list “aux” (i.e., among the nodes which are not yet covered). For doing that one can use any Steiner tree heuristic. In our simulations we use the minimum spanning tree (MST) heuristic for simplicity.

65.3.3.2 Distributed Variant

The previous algorithm may be useful for some kind of networks. However, in general, a distributed algorithm is preferred for wireless ad hoc networks. Hence, Ruiz and Gomez-Skarmeta [23] proposed the distributed approach described below.

The previous protocol consists of two different parts: (i) construction of cost-efficient subtrees and (ii) building a Steiner tree among the roots of the subtrees.

To build a Steiner tree among the roots of the subtrees, Ruiz assumed the utilization of the MST heuristic in the previous protocol. This is a centralized heuristic consisting of two different phases. First, the algorithm builds the metric closure for the receivers on the whole graph, and then, an MST is computed on the metric closure. Finally, each edge in the MST is substituted by the shortest path (in the original graph) between the two nodes connected by that edge. Unfortunately, the metric closure of a graph is hard to build in a distributed way. Thus, Ruiz approximated the MST heuristic with the simple, yet powerful, algorithm presented in Algorithm 65.2. The source, or the root of the subtree in which the source is (called source-root), will start flooding a route request message (RREQ). Intermediate nodes, when propagating that message will increase the hop count. When the RREQ is received by a root of a subtree, it sends a route reply (RREP) back through the path which reported the lowest hop count. Those nodes in that path are selected as multicast forwarders (MF). In addition, a root of a subtree, when propagating the RREQ will reset the hop count field. This is what makes the process very similar to the computation of the MST on the metric closure. In fact, we achieve the same effect, which is that each root of the subtrees, will add to the Steiner tree the path from itself to the source-root, or the nearest root of a subtree. In the case in which two neighboring nodes are far away from S but at the same hop count, the node-ID is used as a tie-breaker.

Algorithm 65.2 Distributed approximation of MST heuristic MNT2

```
if $thisnode.id = source - root$ then
    Send RREQ with RREQ.hopcount=0
end if
if rcvd non duplicate RREQ with better hopcount then
    prevhop ← RREQ.sender
    RREP.nexthop ← prevhop
    RREQ.sender ← $thisnode.id$
    if $thisnode.isroot$ then
        send(RREP)
        RREQ.hopcount ← 0
    else
        RREQ.hopcount++;
    end if
    send(RREQ)
end if
if received RREP and RREP.nexthop = $thisnode.id$ then
    Activate MF_FLAG
    RREP.nexthop ← prevhop
    send(RREP)
end if
```

This avoids a deadlock by preventing each of them from calling the other as its selected next hop. The one with the lowest ID will always select the other. This mechanism and the way in which the algorithm is executed from the source-root to the other nodes guarantees that the obtained tree is connected.

The second part of the algorithm to make distributed is the creation of the cost-effective subtrees. However, this part is much simpler and it can be done locally with just a few messages. Receivers flood a Subtree_Join (ST_JOIN) message only to its 1-hop neighbors indicating the multicast group to join. These neighbors answer with a Subtree_Join_Ack (ST_ACK) indicating the number of receivers they cover. This information is known locally by counting the number of (ST_JOIN) messages received. Finally, receivers send again a Subtree_Join_Activation (ST_JOIN_ACT) message including their selected root, which is the neighbor which covers the greatest number of receivers. This is also known locally from the information in the (ST_ACK). Those nodes which are selected by any receiver, repeat the process acting as receivers. Nodes which already selected a root do not answer this time to ST_JOIN messages.

65.3.4 Performance Evaluation

Ruiz and Gomez-Skarmeta [23] presented some performance evaluation of their approach. We have reproduced their simulations with the same parameters except that the number of nodes has been fixed at 600, and the area has been ranged from 750 × 750 to 2250 × 2250 m^2. The simulated algorithms are the two minimum bandwidth algorithms (MNT and MNT2 respectively) as well as the MST heuristic to approximate Steiner trees. In addition, we also simulated the SPT algorithm, which is the one which is used by most multihop multicast routing protocols proposed to date.

Performance metrics are also the same as Ruiz and Gomez-Skarmeta used in Ref. [23] and the reader can refer there for details. The results shown correspond to the 1250 × 1250 m^2 area and 150 receivers. For each combination of simulation parameters, a total of 91 simulation runs with different randomly generated graphs were performed. The error in the graphs shown below are obtained using a 95% confidence level.

65.3.4.1 Performance Analysis

In Figure 65.2, we show for a network with an intermediate density (600 nodes in 1250 × 1250 m^2 area) how the number of transmissions required varies with respect to the number of receivers. For a low number of them, the minimum bandwidth schemes do not offer significant differences compared to the Steiner tree heuristic. This is because receivers tend to be very sparse and it is less likely that cost-effective trees are built. However, as the number of receivers increases, the creation of cost-effective trees is favored, making

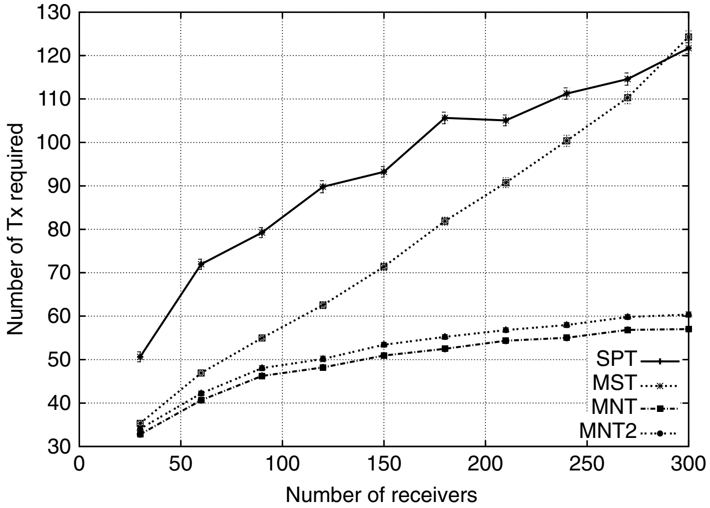


FIGURE 65.2 Number of Tx at increasing number of receivers.

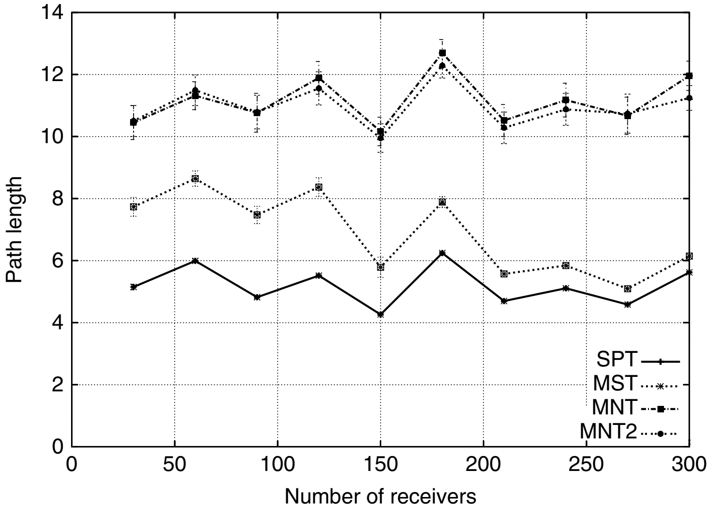


FIGURE 65.3 Mean path length at increasing number of receivers.

the MNT and MNT2 algorithms achieve significant reductions in the number of transmissions required. In addition, given that the SPT approach does not aim at minimizing the cost of the trees, it shows a lower performance compared to any of the other approaches. The distributed MNT2 algorithm, by not using the metric closure, gets a slightly lower performance compared to the centralized approach. However, both of them have very similar performance, which allow them to offer substantial bandwidth savings compared to the Steiner tree (i.e., MST heuristic).

In Figure 65.3 we represent the mean path length. MNT and MNT2 offer a higher mean path length because grouping paths for several receiver require deviating from their shortest paths for some of the receivers. As we can see, this metric is much more variable to the number of receivers than the number of transmissions was for the heuristic approaches. This is why the error bars are reporting a larger confidence interval for MST, MNT, and MNT2.

In Figure 65.4 we present the variation of the number of transmissions as the density varies, for 150 receivers. As the figure depicts, the higher the density, the better is the performance of all the approaches.

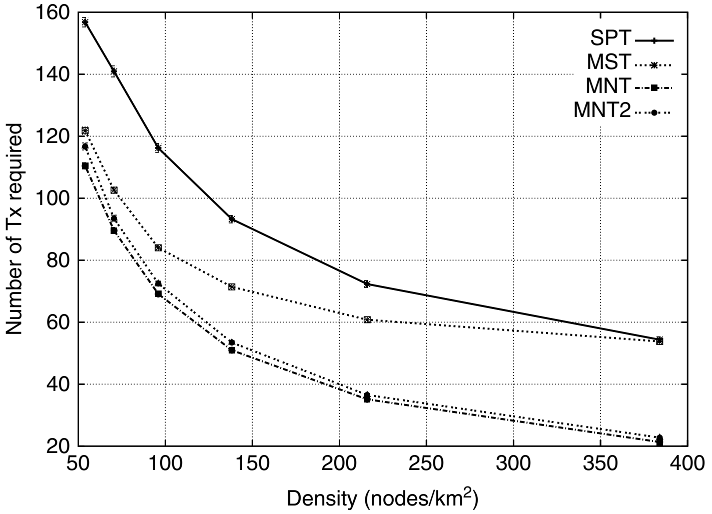


FIGURE 65.4 Number of Tx with varying network density for 150 receivers.

The reason is that higher densities imply shorter path length (note that number of nodes is fixed). So, in general, one can reach the receivers with less number of transmissions regardless of the routing scheme. However, if we compare the performance across approaches, we can see that the reduction in the number of transmissions achieved by MNT and MNT2 is higher as the density increases. This can be easily explained by the fact that for higher densities it is more likely that several receivers can be close to the same node, which facilitates the creation of cost-effective subtrees.

We can observe that the number of receivers has small impact on the performance comparison compared to that of the density of the network.

65.4 Cost-Efficient Geographic Multicast Routing

Routing in sensor networks differ from routing in ad hoc networks. Position information is almost intrinsic to WSNs. In fact, measurements provided by sensor nodes do not usually have proper meaning unless the geographical information regarding that sensor is also reported. Position information, if available to each sensor, also simplifies routing task. This approach is commonly known as "geographic routing." A survey of position-based routing schemes is given in Ref. [25].

The general idea is very simple. Given a source s and a destination d (the destination is normally a fixed sink whose location is known to all sensors), the source s sends the data packet to be delivered to the neighbor which is closest to the destination. This neighbor will repeat the process again, until the message is eventually delivered to the destination. For the case in which the algorithm reaches a local minima (i.e., there is no neighbor making progress toward the destination), Bose et al. [26] proposed a recovery scheme called "Face routing" which guarantees delivery.

Similar to ad hoc network routing, different metrics can be minimized when finding the route toward a destination using geographic routing. The most used one is the reduction of the hop count. However, there are also proposals to reduce the power consumption [27], delay, etc. Stojmenovic [28] proposed a general framework to optimize different metrics. The rest of the chapter will be devoted to the explanation of that framework, and its application to the problem of efficient geographic multicast routing.

65.4.1 The Cost over Progress Framework

A frequent solution when dealing with multiple metrics in geographic routing in the literature is to introduce additional parameters, not present in the problem formulation, as part of the solution protocol. For instance, the neighbor selection function is changed to something like

$$\alpha(metric_1) + \beta(metric_2) + \cdots + \omega(metric_n)$$

where Greek letters are the additional parameters considered and $metric_i$ the different metrics (e.g., delay, hop count, power consumption, etc.).

So, the performance of such a protocol often depends on the particular values for the set of parameters. In most cases, the optimal values for these parameters depend on global network conditions, which may be beyond the knowledge available to tiny sensors. In other cases, the computational and communication cost required to obtain such information is higher than the benefits provided by the particular protocol. Another typical approach is to use thresholds for the solutions, which has the effect of eliminating certain options in the protocols which may lead to suboptimal solutions or to failures.

To avoid those issues, a simple but elegant solution is to use as the neighbor selection function, a cost over progress ratio. The idea is to optimize the ratio of operation costs (in terms of the particular metrics considered in the problem statement) and progress made (e.g., reduction in distance to the destination in the case of routing). For network coverage problems, the same concept can be applied by redefining the cost function, and considering an appropriate progress function (e.g., additional area covered). To better understand this idea, we will give a couple of examples of its application.

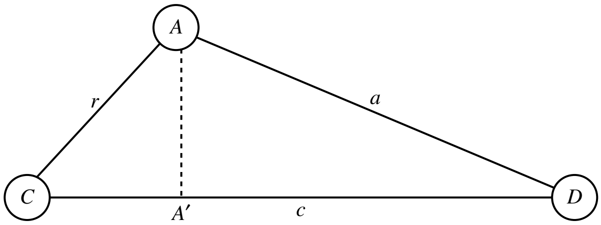


FIGURE 65.5 Best neighbor selection in localized routing schemes.

Example 65.1

We consider the case in which we want to provide geographic routing with minimal power consumption. We consider Figure 65.5 as a reference, where C is the source, D the destination, and A a candidate neighbor and A' the projection of A on $\overline{CD}$. In addition, we have that $|\overline{CD}| = c$, $|\overline{AD}| = a$, and $|\overline{CA}| = r$.

The power needed to send a message from C to A is proportional to $r^\alpha + k$, where α is the power attenuation factor ($2 \leq \alpha \leq 6$), while k is a constant ($k > 0$) that accounts for running circuits at the transmitter and receiver nodes. In our framework, this power can be used as the cost measure. The progress can be measured according to Figure 65.5 as $c - a$. Therefore, the neighbor that minimizes $(r^\alpha + k)/(c - a)$ is the one to be selected.

In the next section, we discuss how the same framework can be applied to the geographic multicast routing problem.

65.4.2 Geographic Multicasting with Cost over Progress

We now focus on the multicasting problem represented in Figure 65.6. A source node C wishes to send a packet to several destinations (sinks) with known positions $(D_1, \ldots, D_n)$. It is assumed that the number of such destinations is small, which is reasonable for the scenario in which a sensor reports to several sinks.

Mauve et al. [29] proposed a geographic multicast protocol which considers the total hop count as the metric to optimize, and distances from neighbors to destinations as part of the criterion to optimize. The impact of each of these aspects in the final neighbor selection is controlled by an external parameter (λ) whose best value is to be separately determined. We describe below how the same problem can be solved with the cost over ratio framework, without the need for such additional parameters.

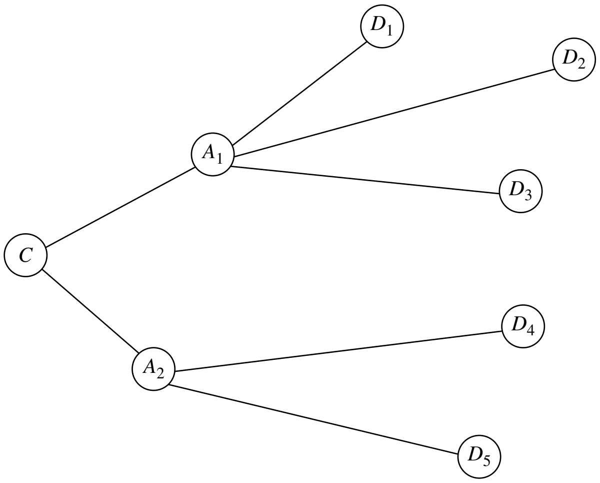


FIGURE 65.6 Evaluating candidate forwarding set from neighbors.

Let us assume a node C, after receiving a multicast message including the positions of the destinations $D_1, D_2, \ldots, D_k$. And let us also assume that C evaluates neighbors $A_1, A_2, \ldots, A_m$ for forwarding. If there is one neighbor which is closer to all destinations than C, then it may happen that there is only one next hop selected. However, it may also happen that the multicast routing task needs to be split across multiple neighbors, each handling a subset of destinations.

Consider the case in Figure 65.6 as an illustration of the general principle. The current total distance to deliver the message from C to all receivers is $T_1 = |\overline{CD_1}| + |\overline{CD_2}| + |\overline{CD_3}| + |\overline{CD_4}| + |\overline{CD_5}|$. If C now considers neighbors A_1 and A_2 as forwarding nodes covering D_1, D_2, D_3 and D_4, D_5, respectively, the new total remaining distance would be $T_2 = |\overline{A_1D_1}| + |\overline{A_1D_2}| + |\overline{A_1D_3}| + |\overline{A_2D_4}| + |\overline{A_2D_5}|$, and the "progress" made is $T_1 - T_2$. The "cost" is the number of selected neighbors (i.e., two in the previous figure). Thus, the forwarding set $\{A_1, A_2\}$ is evaluated as $2/(T_1 - T_2)$. So, among all candidate forwarding sets, the one with optimal value of this expression is selected. Those destinations for which there is no neighbor closer to them, will be reached using Greedy Face Greedy (GFG) routing [26] directly to them. Note that the number of expressions to evaluate grows with number of neighbors and number of destinations. We provide below an enhanced algorithm to reduce the number of evaluations.

65.4.2.1 Exhaustive Enumeration by Set Partitioning

Given k destinations, the algorithm can consider all S_k partitions. For each set given in the set partition, check whether there is a node which is closer to the destinations in the set than the current node C. If it is not possible to find such a node for a set, that particular partition is ignored. For those being possible, the cost/progress ratio is computed, and the best one is selected. This solution is applicable for small number of destinations (e.g., up to 5). For a larger number, it becomes exponential in k, and therefore a faster greedy solution is needed. A fast algorithm for generating set partitions is given in Ref. [30].

65.4.2.2 Greedy Selection of Set Partitions

The goal of this greedy selection of set partitions is to reduce the number of partitions (destination sets) being evaluated. The destinations for which there is no closer neighbor are served using the GFG protocol [26]. Thus, we start with the set of destinations $\{D_1, D_2, \ldots, D_k\}$, for which there is at least one node closer to them than C.

For each of these D_i we choose the best neighbor A_i of C as if it was the only destination. If there are several destinations for which the best neighbor is the same, then we merge those destinations into a single set $\{M_i\}$. At the end of this phase, we have an initial set partition of the destinations $\{M_1, M_2, \ldots, M_l\}$, and there is a different neighbor A_i for each subset M_i. Each M_i has its cost over progress $1/P_i$, and the whole partition also has its overall cost over progress being equal to $l/(P_1 + P_2 + \cdots + P_l)$. In the example of Figure 65.6, there are two subsets $M_1 = \{D_1, D_2, D_3\}$ and $M_2 = \{D_4, D_5\}$. The cost over progress of M_1 is $1/P_1$, where $P_1 = (|\overline{CD_1}| + |\overline{CD_2}| + |\overline{CD_3}|) - (|\overline{A_1D_1}| + |\overline{A_1D_2}| + |\overline{A_1D_3}|)$. Similarly, the cost over progress for M_2 is $1/P_2$, where being $P_2 = (|\overline{CD_4}| + |\overline{CD_5}|) - (|\overline{A_2D_4}| + |\overline{A_2D_5}|)$. The overall cost over progress ratio of the whole partition is then $2/(P_1 + P_2)$.

After this first iteration, we will repeatedly try to improve the cost over progress ratio, until this was not possible in a given iteration. In each iteration, we try to merge each pair M_i, M_j to see whether they can improve the cost–progress ratio by merging into one set. This merge is done selecting a new A_j, being the neighbor of C which is closer than C to all destinations in $M_i \cup M_j$ and provides best ratio. If such A_j does not exist, merge is not possible. From all possible merges which improve the ratio compared to the partition set, select the one with higher ratio, and update the set partition by effectively merging those sets, and decreasing l by 1. The next iteration starts with this new set partition. The process is repeated until no new merging improves the ratio.

It is easy to prove that this algorithm would test $O(k^3)$ cases rather than k!

The described algorithm is based on position information. If it is not available, it can be applied on a version where distances between nodes are replaced by hop counts between them. Each receiver may flood the network, so that each node may learn hop count distances to all receivers. These distances can be used in the described protocol to determine multicast routes.

65.5 Discussion

Multicast routing has proven to be an interesting problem requiring approximation algorithms, and simple yet efficient heuristic solutions. As we have seen, multicast routing problems are NP-complete even when the metrics to optimize are not very complicated.

In this chapter, we have shown examples of such greedy algorithms and their applicability to the multicast routing problem in two different environments: wireless ad hoc networks, and WSNs.

In ad hoc networks, we have shown that the traditional Steiner tree problem does not guarantee optimality regarding the overall bandwidth consumption of the multicast tree, and we have applied a heuristic epidemic algorithm to approximate efficiently those bandwidth-efficient trees.

For sensor network scenarios, we have shown how the general geographic routing concept can be extended to solve the multicast routing task in a localized way. In addition, we have explained how the general cost–progress ratio framework can be also used for such problem. As shown here, using this framework we avoid adding extra parameters to the problem formulation, which is one of the issues present in most of the existing solutions in the literature.

We believe that the concepts presented here will be useful for solving a number of other network layer problems, or even variants of these ones based on different assumptions or optimality criteria.

Acknowledgments

Work supported by the "Ramon y Cajal" work programme from the Spanish Education Ministry.

References

[1] Deering, S., Multicast Routing in a Datagram Internetwork, Ph.D. thesis, Electrical Engineering Department, Stanford University, 1991.

[2] Deering S. and Cheriton, D. R., Multicast routing in datagram internetworks and extended LANs, *Trans. Comput. Syst.,* 8(2), 85, 1990.

[3] Moy, J., Multicast routing extensions for OSPF, *CACM,* 37(8), 61, 1994.

[4] Ballardie, T., Francis P., and Crowcroft, J., Core Based Trees (CBT)—an architecture for scalable inter-domain multicast routing, *Proc. of ACM SIGCOMM,* 1993, p. 85.

[5] Deering, S, Estrin, D., Farinacci, D., Jacobson, V., Liu, C. G., and Wei, L., The PIM architecture for wide-area multicast routing, *IEEE/ACM Trans. Networking,* 4(2), 153, 1996.

[6] Wu, C. W., Tay Y. C., and Toh, C. K., Ad hoc multicast routing protocol utilizing increasing id-numberS (AMRIS) functional specification, Internet-draft, work in progress, draft-ietf-manet-amris-spec-00.txt, 1998.

[7] Belding-Royer, E. M. and Perkins, C. E., Multicast operation of the ad-hoc on-demand distance vector routing protocol, *Proc. of ACM/IEEE MOBICOM,* 1999, p. 207.

[8] Ji, L. and Corson, S., A lightweight adaptive multicast algorithm, *Proc. of IEEE GLOBECOM,* 1998, p. 1036.

[9] Jetcheva, J. G. and Johnson, D. B., Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks, *Proc. of ACM MobiHoc,* 2001, p. 33.

[10] Toh, C. K., Guichala, G., and Bunchua, S., ABAM: On-demand associativity-based multicast routing for mobile ad hoc networks, *Proc. of IEEE VTC,* 2000, 987.

[11] Lee, S. J., Su, W., and Gerla, M., On-demand multicast routing protocol in multihop wireless mobile networks, *ACM/Kluwer Mobile Networks Appl.,* 7(6), 441, 2002.

[12] Garcia-Luna-Aceves, J. J. and Madruga, E. L., The core-assisted mesh protocol, *IEEE J. Selected Areas Commun.,* 17(8), 1380, 1999.

[13] Bommaiah, E., Liu, M., MacAuley A., and Talpade, R., AMRoute: ad hoc multicast routing protocol, Internet-draft, work in progress, draft-talpade-manet-amroute-00.txt, 1998.

[14] Sinha, P., Sivakumar, R., and Bharghavan, V., MCEDAR: multicast core-extraction distributed ad hoc routing, *Proc. of Wireless Communication and Networking Conf.*, 1999, p. 1313.

[15] Ji, L. and Corson, M. S., Differential destination multicast: a MANET multicast routing protocol of small groups, *Proc. of INFOCOM*, 2001, p. 1192.

[16] Ruiz, P. M. and Gomez-Skarmeta A. F., Mobility-aware mesh construction algorithm for low data-overhead multicast ad hoc routing, *J. Commun. Networks*, 6(4), 331, 2004.

[17] Karp, R. M., Reducibility among combinatorial problems, *In Complexity of computer computations*, Plenum Press, New York, 1975, p. 85.

[18] Waxman, B. M., Routing of multipoint connections, *IEEE J. Selected Areas Commun.*, 6(9), 1617, 1998.

[19] Kou, L., Markowsky, G., and Berman L., A fast algorithm for Steiner trees, *Acta Informatica*, 2(15), 141, 1981.

[20] Plesnik, J., The complexity of designing a network with minimum diameter, *Networks*, 11, 77, 1981.

[21] Zelikovsky, A., An 11/6-approximation algorithm for the network Steiner problem, *Algorithmica*, 9, 463, 1993.

[22] Rajagopalan, S. and Vazirani, V. V., On the bidirected cut relaxation for the metric Steiner tree problem, *Proc. of SODA*, 1999, p. 742.

[23] Ruiz, P. M. and Gomez-Skarmeta A. F., Approximating optimal multicast trees in wireless multihop networks, in *Proc. Symp. on Computers and Communications*, 2005, p. 686.

[24] Clark, B. N., Colbourn, C. J., and Johnson, D. S., Unit disk graphs, *Discrete Math.*, 86, 165, 1990.

[25] Giordano, S. and Stojmenovic, I., Position based routing in ad hoc networks, a taxonomy, in *Ad Hoc Wireless Networking*, Cheng, X., Huang, X., and Du, D. Z., Eds., Kluwer, Dordrecht, 2003.

[26] Bose, P., Morin, P., Stojmenovic, I., and Urrutia, J., Routing with guaranteed delivery in ad hoc wireless networks, *ACM Wireless Networks*, 7(6), 609, 2001.

[27] Stojmenovic, I. and Lin, X., Power aware localized routing in wireless networks, *IEEE Trans. Parallel Distributed Syst.*, 12(11), 1122, 2001.

[28] Stojmenovic, I., Localized network layer protocols in sensor networks based on optimizing cost over progress ratio and avoiding parameters, *IEEE Network*, 20(1), 21–27, 2006.

[29] Mauve, M., F*üβ*ler, H., Widmer, J., and Lang, T., Position-Based Multicast Routing for Mobile Ad-Hoc Networks, TR-03-004, Department of CS, University of Mannheim, 2003.

[30] Djokic, B., Miyakawa, M., Sekiguchi, S., Semba, I., and Stojmenovic, I., A fast iterative algorithm for generating set partitions, *Comput. J.*, 32(3), 281, 1989.

66

Approximation Algorithm for Clustering in Ad Hoc Networks

Lan Wang
Old Dominion University

Stephan Olariu
Old Dominion University

66.1 Introduction

Since flat networks do not scale, it is a time-honored strategy to overlay a virtual infrastructure on a physical network. There are, essentially, two approaches to doing this. The first approach is protocol-driven and involves crafting a virtual infrastructure in support of whatever protocol happens to be of immediate interest. While the resulting virtual infrastructure is likely to serve the protocol well, more often than not, the infrastructure is not useful for other purposes. This is unfortunate, as its consequence is that a new infrastructure has to be invented and installed from scratch for each individual protocol in a given suite. In bandwidth-constraint Mobile Adhoc NETworks (MANET), maintaining different virtual infrastructures for different protocols may involve excessive overhead. The alternate approach is to design the virtual infrastructure with no particular protocol in mind. The challenge, of course, is to design the virtual infrastructure in such a way that it can be leveraged by a *multitude* of different protocols. Such a virtual infrastructure is called *general-purpose* as opposed to *special-purpose* if it is designed in support of just one protocol. The benefits of a general-purpose virtual infrastructure are obvious.

The important problem of identifying general-purpose infrastructures that can be leveraged by a multitude of different protocols has not yet been addressed in MANET. We view the main contribution of this work as the first step in this direction. Specifically, we identify *clustering* as the archetypal candidate for establishing a general-purpose virtual infrastructure for MANET. Motivated by the idea that a virtual infrastructure having a decent chance of becoming truly general-purpose should be able to make a large MANET appear *smaller* and *less dynamic*, we propose a novel clustering scheme based on a number of properties of *diameter-2* graphs. Compared to virtual infrastructures with central nodes, our virtual infrastructure is more symmetric and stable, but still lightweight. In our clustering scheme, cluster initialization naturally blends into cluster maintenance, showing the unity between these two operations. We call our algorithm *tree-based* since cluster merge and split operations are performed based on a spanning tree maintained at some specific nodes. Extensive simulation results have shown the effectiveness of our clustering scheme when compared to other schemes proposed in the literature.

The remainder of this chapter is organized as follows: following the background of our work described in Section 66.2, Section 66.3 presents technicalities that underlie the tree-based clustering scheme; Section 66.4 provides the details of the tree-based clustering algorithms; Section 66.5 presents our simulation results in terms of several generic metrics; Section 66.6 discusses the application of the clustering scheme as appromiation algorithms in the context of topology control; finally, Section 66.7 offers concluding remarks and directions for further work.

66.2 Motivation

Essentially, a cluster is a subset of the nodes of the underlying network that satisfies a certain property P. At the network initialization stage, a *cluster initialization* algorithm is invoked and the network is partitioned into individual clusters each satisfying property P. Due to node mobility, new links may form and old ones may break, leading to changes in the network topology and, thus, to possible violations of property P. When property P is violated, a *cluster maintenance* algorithm must be invoked. It is intuitively clear that the less stringent property P, the less frequently is cluster maintenance necessary.

The precise definition of the desirable property P of a cluster varies in different contexts. However, there are some general guidelines suggesting instances of P that are desirable in all contexts. One of them is that a *consensus* must be reached quickly in a cluster in order for a cluster to work efficiently. Since the time complexity of the task of reaching a distributed consensus increases with the diameter of the underlying graph [1], *small-diameter* clusters are generally preferred in MANET [2]. As an illustration, some authors define property P such that every node in the cluster is *1-hop* away from every other node, that is, each cluster is a diameter-1 graph [3,20]. A less restrictive widely adopted definition of P is the *dominance property* [4–6], which insists on the existence of a central cluster-head adjacent to all the remaining nodes in the cluster. In the presence of a central node, consensus is reached trivially: indeed, the cluster-head dictates the consensus.

Motivated by the fact that a cluster-head may easily become a traffic bottleneck and a single point of failure in the cluster, and inspired by the instability of the virtual infrastructures maintained by the node-centric clustering schemes, in the clustering scheme proposed by Lin and Gerla [7], although the cluster initialization algorithm used is node-centric with the clusters featuring a central cluster-head, once clusters are constructed [7], eliminates the requirement for a central node, defining the cluster simply as a diameter-2 graph. Only when the cluster is no longer a diameter-2 graph will a cluster change occur. This definition imposes fewer constraints on a cluster and hence may result in significant improvement on the stability of the resulting virtual infrastructure. In addition, Nakano and Olariu [8] have shown that a distributed consensus can be reached fast in a diameter-2 cluster. In the light of these observations, in this work we adopt the *diameter-2 property* as the *defining property of a cluster.*

The basic idea of the *degree*-based *cluster maintenance algorithm* of Ref. [7] is the following: when a violation of the diameter-2 property is detected, the highest degree node and its one-hop neighbors remain in the original cluster and all the other nodes leave the cluster. It is expected that a leaving node will join another cluster or form a new cluster by itself. Unfortunately, the description of the algorithm in Ref. [7] is very succinct and many important details are glossed over.

In fact, there are several problems with the above degree-based cluster maintenance algorithm as discussed in Ref. [7]. To illustrate, consider the cluster topology in Figure 66.1(a). When the link (3,4) is broken due to mobility, the diameter-2 property is violated. One problem is that various nodes have a different local view, precluding them from reaching a global consensus as to which node(s) should leave the cluster. To wit, even if the highest degree of nodes in Figure 66.1(b) is propagated throughout the entire topology, the nodes still do not have sufficient information to decide whether or not they should leave the cluster. For example, node 3 is adjacent to node 2 which has degree two, thus being a highest-degree node. Consequently, node 3 decides that it should not leave the cluster. Likewise, node 5 is adjacent to node 4, which also has the highest degree and decides that it should not leave the cluster. The net effect is that no node will leave, invalidating the correctness of the cluster maintenance algorithm.

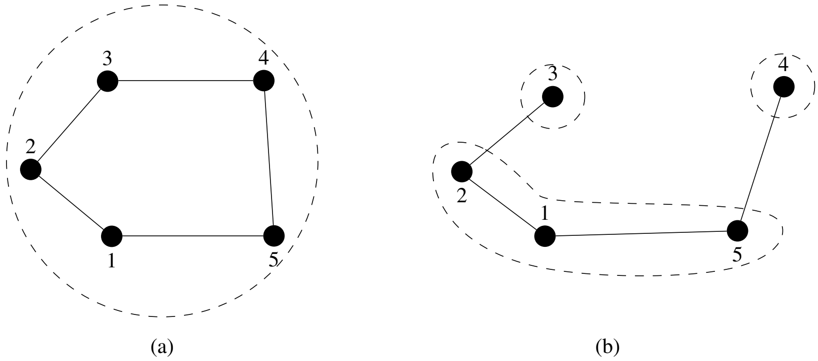


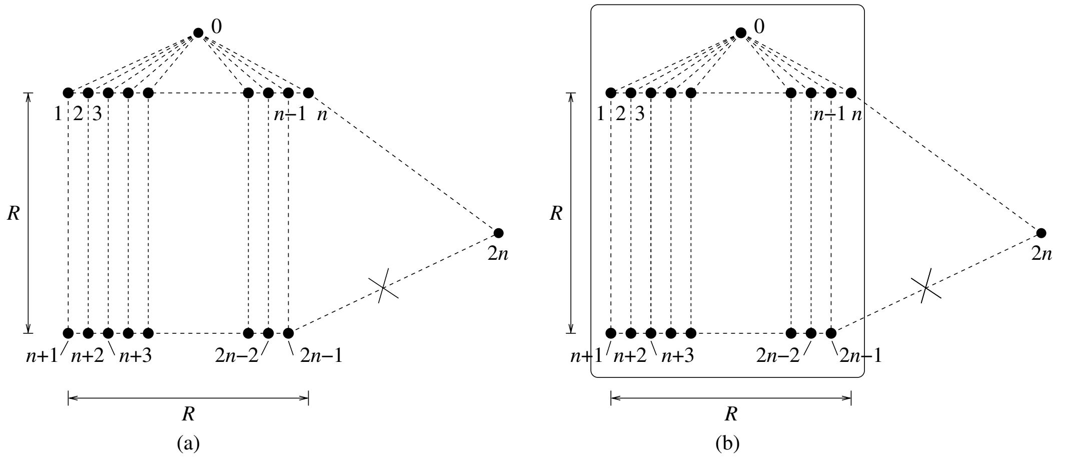
FIGURE 66.1 An example of the degree-based cluster maintenance algorithm.

Notice that the insecurity we just outlined stems, in part, from the fact that in Figure 66.1(b) there are three highest-degree nodes: nodes 1, 2, and 5. The above problem can be helped somewhat by using the lowest nodeID criterion to break ties. Under this criterion, node 1 and its one-hop neighbors, nodes 2 and 5, stay in the original cluster, and nodes 3 and 4 leave. Thus, in this case, the original cluster is partitioned into three clusters: $\{1,2,5\}$, $\{3\}$, and $\{4\}$.

Furthermore, if the cluster maintenance algorithm of Ref. [7] is to be fully distributed, each node must maintain the whole topology of the cluster; otherwise, the nodes cannot reach a consensus as to which is the *unique* node with the highest degree. Note that maintaining the complete topology of the cluster at each member node requires flooding the formation and breakage of *every* link to *all* the other nodes in the cluster, involving a large overhead.

The cluster maintenance algorithm of Ref. [7] tries to minimize the *number of node transitions* between clusters and this number is used to evaluate the *stability* of the cluster infrastructure. However, there is no guarantee that this algorithm will minimize node transitions. In the example shown in Figure 66.2(a), there are $2n+1$ nodes in the cluster, numbered from 1 to $2n+1$. Nodes $1, 2, \ldots, n$ are within transmission range (R) from each other; similarly, the nodes $n+1, n+2, \ldots, 2n-1$ are within transmission range from each other. With the breakage of link between nodes $2n-1$ and $2n$, the cluster is no longer diameter-2. Nodes $1, 2, \ldots, n$ have degree $n+2$ and are the highest-degree nodes. Assume that node 1 is chosen as the maintenance leader. In this case, according to the degree-based algorithm, $n-1$ nodes (namely, nodes $n+2, n+3, \ldots, 2n$) leave the cluster while, in fact, the minimum number of nodes that have to leave the cluster is just one as shown in Figure 66.2(b).

Moreover, using the number of node transitions as the *sole* criterion to assess the goodness of a cluster maintenance algorithm is misleading since: (a) it implicitly assumes that the highest-degree node is the same as the logical cluster representative. This assumption is not attractive since during normal operation of a cluster, the highest-degree node may change frequently due to link changes. If every highest-degree node change results in a migration of the logical cluster representative, a significant amount of overhead will be involved. (b) It assumes that only *leaving* nodes are responsible for the overhead in the cluster maintenance procedure. In reality, during the maintenance procedure, all nodes in the involved clusters participate in computation and message passing for determining the new cluster membership. Consider an example simulation for two clustering schemes 1 and 2. During the simulation, in Scheme 1, a cluster with 100 nodes are split once into two clusters, each with 50 nodes; in Scheme 2, a cluster with 100 nodes decreases its size by one node for 30 times. It is not clear that Scheme 2 is definitely more stable than Scheme 1. (c) In many cases, the degree-based algorithm generates *single-node* clusters. Such a cluster is of little use and must merge with some other existing cluster. This operation should be considered part of the overhead introduced by the cluster maintenance algorithm. Consider the following cluster infrastructure: each node is a single-node cluster and cluster merge never occurs. In such an infrastructure, the number of node transitions is 0. However, this is a very poor cluster infrastructure and the benefits of clustering are



(a) (b)

FIGURE 66.2 An example in which the degree-based algorithm generates a number of leaving nodes.

lost. This example clearly illustrates the tradeoff between cluster *stability* and *quality*. We must consider both metrics when evaluating the performance of a cluster maintenance algorithm.

66.3 Technicalities

The main goal of this section is to develop the graph-theoretic machinery that will be used by our clustering algorithms. As customary, we model a multihop ad hoc network by an undirected graph $G = (V, E)$ in which V is the set of nodes and E is the set of links between nodes. The edge $(u, v) \in E$ exists whenever nodes u and v are one-hop neighbors. Each node in the network is assigned a unique identifier (nodeID). The distance between two nodes is the length of the *shortest* path between them. The diameter of a graph is the largest distance between any pair of nodes. Our cluster maintenance algorithm relies on the following theorems of diameter-d graphs.

Theorem 66.1

Consider a diameter-d graph G and an arbitrary edge e of G. Let $G' = G - e$ be the graph obtained from G by removing edge e. If G' is connected, then there must exist a node in G' whose distance to every other node is at most d. Moreover, the diameter of G' is at most $2d$.

Proof

Assume that the edge $e = (u, v)$ is removed. Since G' is connected, there must exist a shortest path $P'(u, v) : u = x_1, x_2, \ldots, x_k = v$ joining u and v in G'. Consider node $x_{\lceil \frac{k}{2} \rceil}$. Clearly, the distance from $x_{\lceil \frac{k}{2} \rceil}$ to both u and v is unaffected by the removal of the edge $e = (u, v)$. We claim that the distance in G' from $x_{\lceil \frac{k}{2} \rceil}$ to all the remaining nodes is bounded by d. To see this, consider an arbitrary node y in G and let Π be the shortest path in G joining $x_{\lceil \frac{k}{2} \rceil}$ to y. If Π does not use the edge e, then the removal of e does not affect Π. Assume, therefore, that Π involves the edge e. Assume, without loss of generality, that in Π node v is closer to y than u. However, our choice of $x_{\lceil \frac{k}{2} \rceil}$ guarantees that the path consisting of the nodes $x_{\lceil \frac{k}{2} \rceil}, x_{\lceil \frac{k}{2} \rceil + 1}, \ldots, x_{k-1}, v, \ldots, y$ cannot be longer than Π, completing the first part of the claim.

Consider a BFS tree of G' rooted at $x_{\lceil \frac{k}{2} \rceil}$. We just proved that the depth of this tree is bounded by d, confirming that the diameter of G' is, indeed, bounded by $2d$. □

Theorem 66.1 has the following important consequence that lies at the heart of our cluster maintenance algorithm.

Corollary 66.1

Consider a diameter-2 graph G and an edge e of G. Let $G' = G - e$ be the graph obtained from G by removing edge e. If G' is connected, then there must exist at least one node in G' whose distance to every other node is at most two. Furthermore, the diameter of G' is at most four.

Theorem 66.2

Let G be a diameter-d graph, and let x and y be a pair of nodes that achieve the diameter of G. Then the graph $G' = G - \{x\}$ is connected. Furthermore, in G', any BFS tree rooted at y has depth at most d.

Proof

In G, x is a level-d (leaf) node of any BFS tree rooted at y. Hence removing node x does not affect the distance from y to any other node. Thus, G' must be connected, and in G', any BFS tree rooted at y has depth at most d. □

Theorem 66.2 has the following important consequence that will be used in our cluster maintenance algorithm.

Corollary 66.2

Let G be a diameter-2 graph, and let x be a node in G such that there exists at least one node y in G that is not adjacent to x. In the graph $G' = G - \{x\}$, any BFS tree rooted at y has depth at most two.

Theorem 66.3

Consider a graph $G = (V, E)$, disjoint subsets V_1, V_2 of V, and let G' be the subgraph of G induced by $V_1 \cup V_2$,

(1) if the subgraphs of G induced by V_1 and V_2 are diameter-d graphs, and
(2) if for every node x of V_1, the BFS tree of G' rooted at x has depth at most d then G' is a diameter-d graph.

Proof

Consider an arbitrary pair of nodes u, v in G. We need to show that u and v are at distance at most d in G'. Indeed, if $u, v \in V_1$ (resp. V_2), the conclusion is implied by assumption (1). Consequently, we may assume, without loss of generality, that $u \in V_1$ and that $v \in V_2$. By assumption (2), the BFS tree of G' rooted at u has depth at most d, implying that the distance between u and v is bounded by d. This completes the proof of Theorem 66.3. □

We conclude this section by stating some properties of unit-disk diameter-2 graphs.

We have written a program to generate random unit-disk diameter-2 graphs, and in the one million instances of graphs that was generated by the program, all can be dominated by two nodes. This suggests that the probability that a unit-disk diameter-2 graph is dominated by two nodes is very high. On the other hand, we have been able to construct a counterexample that cannot be dominated by two nodes [9]. Further, we have proven that any *unit-disk* diameter-2 graph can be dominated by at most *three* nodes [9]. Note that this is not true for a *general* diameter-2 graph. For a *general* diameter-2 graph, there is no proved constant upper bound on the size of its minimum dominating set (*MDS*). In the examples shown in Ref. [9], there is a *general* diameter-2 graph with 198 nodes and max node degree 16, so its $|MDS| \geq 12$.

66.4 Our Tree-Based Clustering Algorithm

In MANET link failures caused by node mobility can be predicted by the gradual weakening of the radio signal strength. In addition, since mechanical mobility and radio transmission occur at vastly different time scales, multiple link failures can be treated as a series of single-link failures. With this in mind, in this work we adopt the *single-link failure* and *single-node failure* models where either one *link* or one *node* fails at any one time. We also note that the single-node failure model can be used to account for the scenarios where link breakages occur unpredictably.

We make the following two assumptions: (1) a message sent by a node is received correctly by all its neighbors within a finite time, called a *message round*; and (2) the cluster split algorithm is atomic in the sense that no new link/node failure occurs during its execution.

66.4.1 The Tree-Based Cluster Split Algorithm: Single-Link Failure

In this subsection, we discuss the details of our cluster split algorithm in the case where a single-link failure occurs.

When a node detects the formation/breakage of one of its immediate links, it broadcasts a HELLO beaconing message containing its nodeID, clusterID, cluster size, the nodeIDs, and clusterIDs of its one-hop neighbors, as well as the signal strength of each link to its one-hop neighbors. By receiving such beaconing messages, each node u maintains a depth-2 BFS tree $T(u)$ rooted at u itself and containing only the nodes belonging to the same cluster as u. Clearly, as long as the diameter-2 property holds, the distance between each pair of nodes is at most two, and the tree $T(u)$ contains *all* the nodes in the cluster. Thus, each node knows the number n of nodes in its own cluster.

In our model, each node monitors the signal strength of the links joining it with its one-hop neighbors. When a generic node u detects that the signal strength of one of its links weakens below a threshold value, it reconstructs $T(u)$. By comparing the size $|T(u)|$ of $T(u)$ with n, node u determines whether all the cluster members are still at most two hops away. If it finds that some member cannot be reached in two hops, it broadcasts a VIOLATION message to all of its one-hop neighbors, identifying the single-link failure causing the violation of diameter-2 property. Each node v receiving a VIOLATION message reconstructs its own tree $T(v)$ and checks whether $|T(v)|$ matches n. If there is a mismatch, the node forwards the VIOLATION message to all its neighbors; otherwise, it declares itself a maintenance leader. In other words, a maintenance leader is a node that can reach every other node in at most two hops. By Corollary 66.1, after being forwarded at most once, the VIOLATION message will reach a maintenance leader. Note that there might be multiple maintenance leaders: each of them runs an instance of the cluster split algorithm independently. Finally, *the instance which yields the best quality new clusters is adopted.*

For a generic maintenance leader x, the tree $T(x)$ is composed of: (1) node x itself—the root of the tree; (2) level-1 nodes, that is, x's one-hop neighbors in the original cluster; and (3) level-2 nodes, all the remaining nodes in the original cluster.

During the split procedure, there can be several different considerations as to how to split the original cluster. Our motivation is to minimize the number of newly generated clusters when splitting. In addition, by considering link stability during a split, the newly generated clusters tend to be more stable.

Specifically, a generic maintenance leader x performs the following steps:

Step 1. Node x tries to find the minimum number of level-1 nodes to cover all the level-2 nodes. A level-1 node y can cover a level-2 node z if and only if x can reach z through y. This is an instance of the well-known minimum set covering problem and can be solved using the following greedy heuristic [10].

Initially, all level-2 nodes are marked *uncovered*, and all the level-1 nodes constitute the *total level-1 set*. For each node y in the total level-1 set, x calculates the number N_y of uncovered level-2 nodes that can be covered by y. The node y with the largest N_y is deleted from the total level-1 set, added to the *critical level-1 set* and marked as *new leader*. All the N_y level-2 nodes covered by y are marked *covered*. Node x continues the above process until all the level-2 nodes are marked covered. We call the current total level-1 set as *redundant level-1 set*. For each level-2 node z marked covered, x calculates the stability (i.e., signal strength) of the link STA_{zw} between z and every critical level-1 node w. Denote the node w that has the largest STA_{zw} as p. Node x marks w as new member of p.

Step 2. Next, x tries to use the nodes in the critical level-1 set to cover the nodes that are left in the redundant level-1 set. For each node r in the redundant level-1 set, x determines the stability of the link between r

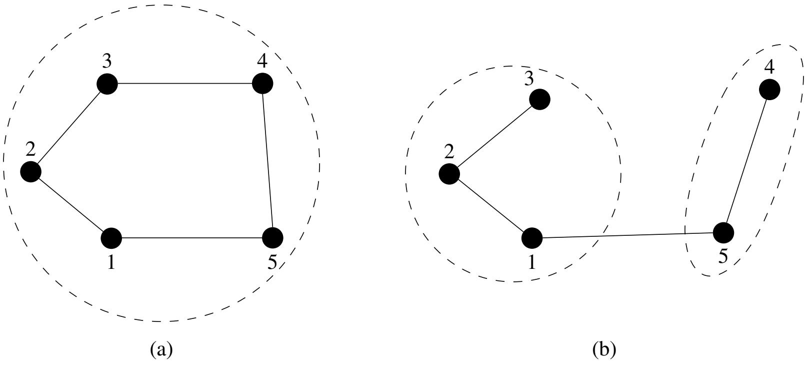


FIGURE 66.3 An example of the tree-based cluster split algorithm.

and each of the critical level-1 nodes adjacent to r. Denote the one that has the most stable link as w; x marks r as a new member of w.

Step 3. x checks whether there exist nodes in the redundant level-1 set. If so, x marks itself new leader and all the uncovered nodes in the redundant level-1 set as new members of x. Otherwise, x finds a new leader q which has the largest link stability value in the critical level-1 set and marks itself as new member of q.

At this point, x has reached its cluster split decision. It broadcasts the result through a MAINTENANCE-RESULT message to all its one-hop neighbors. A node finding itself chosen as a new leader further broadcasts a MEMBER-ENLIST message containing its new cluster membership list. Upon receiving such a message, each node in the original cluster knows its new membership. This completes the split procedure in the case of a single-link failure.

We now illustrate the tree-based split algorithm using an example. There are five nodes in the cluster shown in Figure 66.3(a). When the link (3,4) is broken, nodes 3 and 4 detect that the diameter-2 property is violated. Each of them broadcasts a VIOLATION message. Upon receiving the VIOLATION message, nodes 2 and 5 reconstruct their respective BFS trees. Since neither of them can work as maintenance leader, they forward the VIOLATION message. When node 1 receives the VIOLATION message, it reconstructs $T(1)$ and finds that $|T(1)| = 5$. At this point, node 1 knows that it is a maintenance leader. In $T(1)$, node 2 covers node 3, and node 5 covers node 4. Hence nodes 2 and 5 are chosen as critical level-1 nodes. Assuming that link (1,2) is more stable than link (1,5), node 1 chooses to be covered by node 2. The result of this split procedure is two new clusters: $\{1,2,3\}$ and $\{4,5\}$, as shown in Figure 66.3(b).

66.4.2 The Tree-Based Cluster Split Algorithm: Single-Node Failure

Our cluster split algorithm for the case when a single-node failure occurs relies, in part, on Corollary 66.2. Indeed, by Corollary 66.2, when a single-node failure occurs in a cluster and the tree maintained by the failed node (just before its failure) has depth two, then the resulting graph is still connected (although it need not be diameter-2) and there must be some node that still maintains a BFS tree with depth at most two. This means that a maintenance leader still exists, and that we can still use our tree-based cluster split algorithm. Specifically, when a node detects the *sudden* breakage of a link to/from a one-hop neighbor, it assumes a node failure, deletes the failed node from its cluster membership list, and reconstructs the BFS tree. A VIOLATION message is sent out when necessary, identifying the single-node failure causing the violation of diameter-2 property. The remaining steps are the same as those described in Subsection 66.4.1.

However, if the failed node maintains a depth-1 (as opposed to depth-2) tree before its failure, it is possible that none of the remaining nodes can play the role of maintenance leader. To solve this problem, during the cluster's normal operation phase, when a node finds that it is the only node maintaining a depth-1 tree in the cluster, it periodically runs a MDS algorithm (using a greedy algorithm similar to that described in Subsection 66.4.1) on its one-hop neighbors, and notifies the nodes in the MDS to become candidate maintenance leaders. When the node fails, each candidate maintenance leader detects this failure and immediately broadcasts a MEMBER-ENLIST message containing its new cluster membership list. Upon receiving such a message, each node in the original cluster knows its new membership. This completes the split procedure in the case of a single-node failure.

66.4.3 Merging Clusters

The previous discussion focused on one aspect of cluster maintenance: the cluster split procedure. Clearly, cluster maintenance cannot rely on cluster splitting only, for otherwise the size of the clusters will continually decrease, and we would end up with many one-node clusters, defeating the purpose of clustering. To prevent this phenomenon from occurring, the other necessary component is a mechanism for merging two clusters. The main goal of this subsection is to discuss a simple tree-based cluster merge procedure.

When the members of two clusters move close so that they can reach each other in two hops, the two clusters may be merged. To better control the cluster merge procedure and to prevent it from being invoked too frequently, we introduce the concept of *desirable size of a cluster*. Specifically, given system parameters—desirable cluster size k and tolerances α, β, we insist that clusters should have size in the range $[k - \alpha, k + \beta]$. Clusters of size less than $k - \alpha$ are said to be *deficient*. Only deficient clusters are seeking neighboring clusters with which to merge.

For definiteness, consider a deficient cluster A of size $|A| < k - \alpha$. By receiving HELLO beaconing messages described in Subsection 66.4.1, the nodes in A maintain a list of feasible clusters for merging. Among these, the one, say, B such that $|A| \leq |B|$ and $|A| + |B|$ is as close as possible to k but not exceeding $k + \beta$ is selected. *ClusterID* is used to break ties. Upon selecting B as a candidate, the nodes of A that have a one-hop neighbor in B broadcast a MERGE-REQUEST message. If B is not involved in a merge operation, the nodes of B that have received the MERGE-REQUEST message send back a MERGE-ACK message. At this point, every node in cluster A computes its BFS tree involving nodes in $A \cup B$. A node in A for which the size of the corresponding tree differs from $|A| + |B|$ sends a VIOLATION message to the other nodes in A. By virtue of Property 66.3, if no VIOLATION message is received, $A \cup B$ is a diameter-2 graph. In this case, the nodes in cluster A broadcast a MERGE-CONFIRMATION message to cluster B indicating the new cluster membership and the merge procedure terminates. If, however, a VIOLATION message was received, the merge operation is aborted, a MERGE-ABORT message is sent to the nodes of cluster B, and a new candidate for merging is examined.

We note that the merge operation takes precedence over split. To explain the intuition behind this design decision refer to Figure 66.4. Here cluster X consisting of nodes e and f attempts to merge with cluster Y consisting of nodes a, b, c, and d. Assume that either while the request to merge is issued or just prior to that the edge (a, d) broke, invalidating Y as a cluster. Rather than proceeding with the split operation, as would normally be the case, the merge operation is given priority. As illustrated in the figure, all nodes in X and Y detect that $X \cup Y$ has diameter 2 and is, therefore, a valid cluster. We note, however, that had

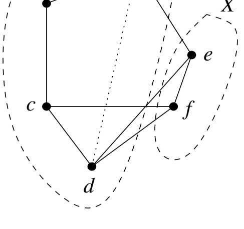


FIGURE 66.4 Illustrating the priority of merge over split.

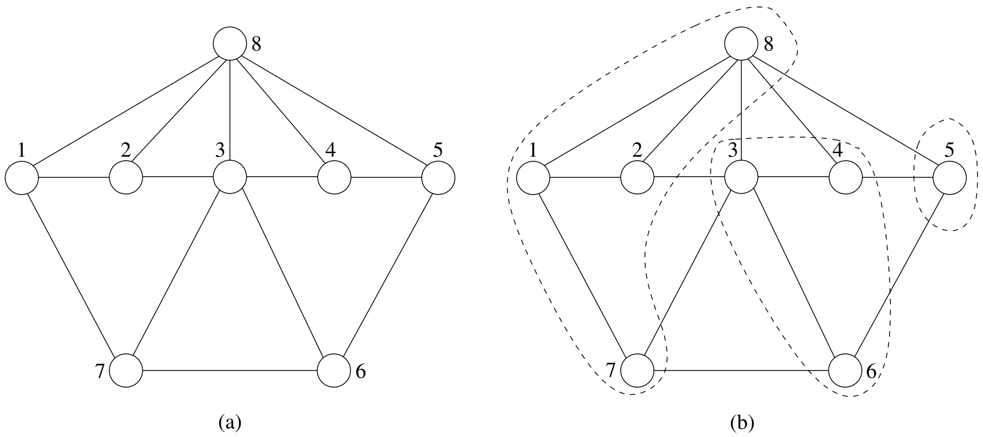


FIGURE 66.5 An example of the cluster initialization algorithm.

$X \cup Y$ had diameter larger than 2, the merge operation would have failed and the nodes in Y would have proceeded with the split operation.

66.4.4 Cluster Initialization

The cluster merge algorithm described in Subsection 66.4.3 is perfectly general and can, in fact, be used for the purpose of cluster initialization. Initially, each *node* is in a *cluster* by itself. The cluster merge algorithm is started as described above. The initialization algorithm naturally blends into cluster maintenance as more and more clusters reach desirable size.

It is worth noting that our cluster initialization algorithm has a number of advantages over the nodeID-based algorithms. First, our algorithm is cluster-centric, as opposed to node-centric. Second, the natural blend of cluster initialization and cluster maintenance shows the unity between these two operations. This is certainly not the case in the vast majority of clustering papers in the literature. Third, our cluster initialization algorithm (just as the cluster merge) can be performed in the presence of node mobility.

Last, our initialization algorithm results in better quality clusters than the nodeID-based algorithms. To see this, consider the subnetwork in Figure 66.5(a) and assume that the desirable cluster size (k) is seven with tolerances $\alpha = \beta = 2$. It is not hard to see that our initialization algorithm actually returns the entire subnetwork as a single cluster—for this graph is diameter-2. On the other hand, the nodeID-based algorithm results in many deficient clusters, as illustrated in Figure 66.5(b).

66.5 Performance Analysis and Simulation Results

In this section, we use simulation to demonstrate the effectiveness of our tree-based clustering scheme compared to other clustering schemes in the literature. We choose LCC [11] as a representative of the node-centric clustering schemes since it avoids the global rippling effect and greatly reduces cluster changes compared to the other nodeID-based algorithms. In addition, it is shown in Ref. [12] that in the unit-disk graph model, LCC is asymptotically optimal with respect to the number of clusters maintained in the system.

66.5.1 Performance Metrics

As discussed in Section 66.2, we need to consider both cluster *quality* and cluster *stability* in our comparison. The number of clusters in the system is generally considered as a good indication of the quality of a cluster infrastructure [12,13]. A clustering scheme that generates and maintains fewer clusters is potentially able to accommodate more nodes in a cluster, hence providing better load balancing among clusters. In our simulation, we count the number of clusters in the system once every second of simulation time. We calculate the sum of these numbers divided by the total simulation time, and we use this *average number of clusters* maintained in the system to characterize cluster *quality*. We note, however, that the number of clusters maintained does not tell the whole story. Given two clustering algorithms that maintain, essentially, the same number of clusters, we prefer the one that generates clusters of roughly equal size to the one that generates a mix of very large and very small clusters. Indeed, clustering schemes that generate very small clusters have to rely on frequent cluster merges to keep cluster quality, clearly an undesirable situation.

To evaluate cluster *stability*, we assume that each cluster chooses one of its member as cluster leader and takes its nodeID as the clusterID. When a node is no longer in the same cluster as its latest cluster leader, this node is considered as a node *changing cluster*. Note that the cluster leader defined here serves only as a reference point that allows us to count and compare the number of *node transitions* in different clustering schemes. In LCC, the central node of a cluster is always the cluster leader. In the diameter-2 schemes, each node initially chooses its nodeID as the clusterID of the single-node cluster. When two clusters merge, the clusterID of the cluster with larger size is used as the new clusterID. When a cluster split happens, among the new clusters, the one which contains the original cluster leader still keeps the original clusterID, and all the other clusters choose the minimum nodeID of its members as the new clusterID. Further, we need to clearly identify the events that can cause cluster changes. In LCC, there are two types of events that can cause nodes to change clusters: (1) a nonleader node is no longer adjacent to its leader; in this case, the node joins another leader, or becomes itself a new leader; (2) when two cluster leaders become neighbors, the one with larger nodeID gives up its role, and all the nodes in its cluster either join a new cluster, or become new leaders by themselves. In the diameter-2 schemes, the two types of events that can cause nodes to change clusters are: (1) a cluster is no longer diameter-2, and is split to several subclusters; (2) a cluster merges with another cluster.

With the above assumptions in mind, we define two measurements to evaluate cluster *stability*: (1) *total number of nodes changing clusters* and (2) *average cluster lifetime*. Specifically, we compare the snapshots of the system taken exactly before and after the execution of the maintenance algorithm triggered by either of the above events. If node *x*'s clusterID after the event is different from its clusterID before the event, then it is counted as a *node changing its cluster*. If a node *x* is a cluster leader before the event, but no longer a leader after the event, then the cluster is considered as disappearing and we stop increasing its lifetime. If a node *x* is not a cluster leader before the event, but becomes one after the event, then a new cluster is considered generated, and we start increasing its lifetime. The *average cluster lifetime* is calculated as the sum of all the cluster lifetimes divided by the number of clusters generated in the simulation.

66.5.2 Simulation Results

We simulate a MANET by placing N nodes within a bounded region of area A. The nodes move according to the random way-point model [14] with zero pause time and constant node speed V. All the nodes have uniform transmission range, which varies from 30 to 210 m in different simulations. For each simulation, we examine the first 300 s of simulation time. All the simulation results presented here are an average of 10 different simulation runs. We also plot 95% confidence intervals for the means. The small confidence intervals show that our simulation results precisely represent the unknown means.

A set of representative simulation results ($N = 100$, $A = 500\,\text{m} \times 500\,\text{m}$, $V = 5\,\text{m/s}$) are shown in Figure 66.6 and Figure 66.7. For the tree-based algorithm, we implement a *baseline* version which does not consider link stability during cluster split. Also, since the tree-based algorithm allows for controlling cluster merging frequency and LCC and the degree-based algorithm do not, we have set the *desirable size of a cluster* to ∞.

(A) *Comparing the node-centric LCC and the cluster-centric diameter-2 schemes.* Figure 66.6(a) indicates that the average number of clusters in the system maintained by the diameter-2 clustering schemes is about *half* of that maintained by LCC. Figure 66.6(b) shows that the number of nodes changing clusters in LCC is significantly larger than in either of the diameter-2 schemes. This is hardly a surprise since LCC is

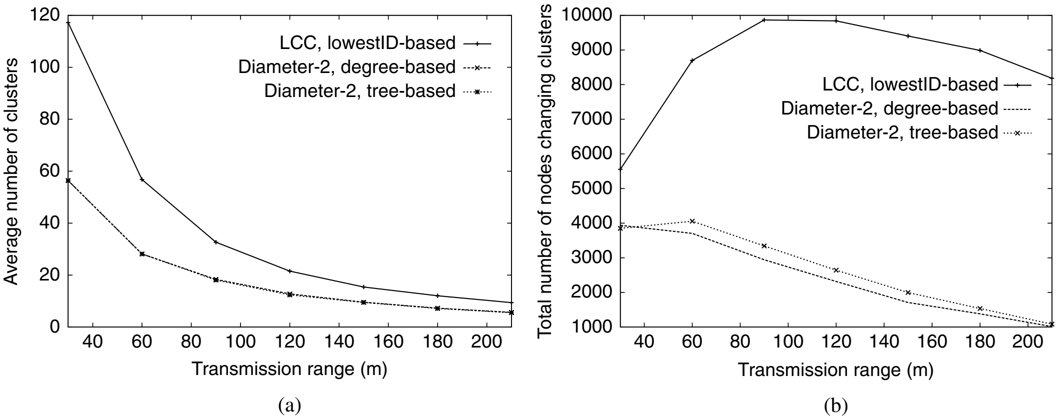


FIGURE 66.6 Comparing the performance of different clustering schemes. (a) Average number of clusters. (b) Total number of nodes changing clusters.

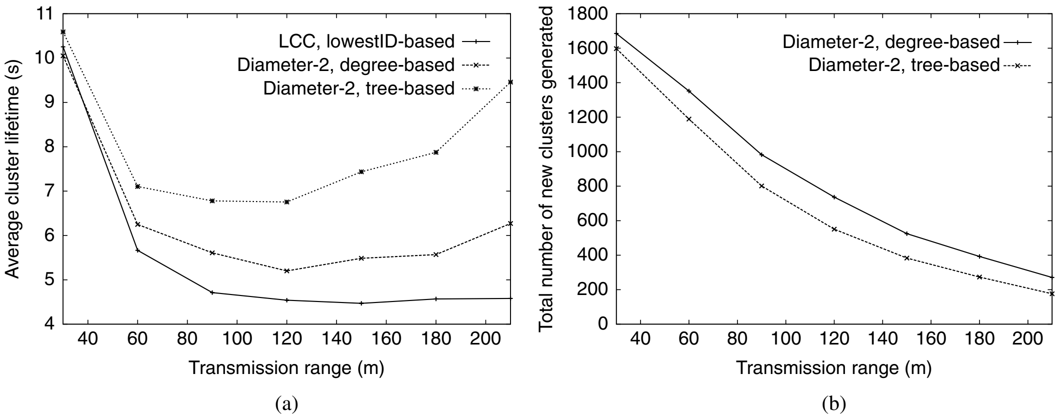


FIGURE 66.7 Comparing the performance of different clustering schemes. (a) Average cluster lifetime. (b) Total number of clusters generated during simulation.

node-centric and it is obvious that clusters predicated on the existence of a central node (the cluster-head) are more brittle than regular diameter-2 clusters. This is further confirmed by Figure 66.7(a) that illustrates that the average lifetime of clusters generated by LCC is shorter than the lifetime of clusters generated by either of the diameter-2 schemes. These results demonstrate that by removing the central-node constraint, the diameter-2 cluster is a much more *stable* structure and can provide *better quality* clusters, especially in MANET applications where central node is not necessary, such as [7,15,16].

(B) *Comparing the tree-based algorithm and the degree-based algorithm.* In terms of the average number of clusters maintained in the system, the tree-based algorithm is slightly better than the degree-based algorithm as shown in Figure 66.6(a). Figure 66.7(a) shows that the average cluster lifetime in the tree-based algorithm is longer than in the degree-based algorithm. From Figure 66.7(b), we can see that the degree-based algorithm generates many more new clusters than the tree-based algorithm. On the other hand, Figure 66.6(b) shows that the total number of nodes changing clusters is significantly larger in the tree-based algorithm than in the degree-based algorithm. The explanation is simple: the degree-based algorithm tends to generate single-node clusters during cluster split, while the clusters generated by the tree-based algorithm are much more *balanced.* The net effect is that when a cluster split/merge happens, a larger number of nodes change clusters in the tree-based algorithm than in the degree-based algorithm.

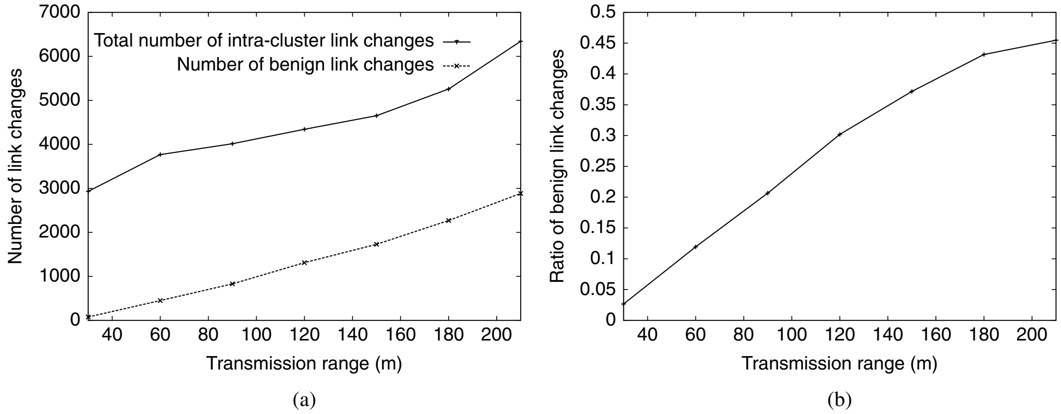


FIGURE 66.8 Comparing the maintenance overhead of tree- and degree-based algorithms. (a) Number of benign intra-cluster link changes. (b) Ratio of benign link changes.

This result shows that the number of nodes changing clusters is not always indicative of the quality of the cluster maintenance algorithm. Note that the single-node clusters generated in the degree-based algorithm are *short-living* and will be merged with other clusters soon, hence they do not significantly influence the *average number of clusters* maintained in the system shown in Figure 66.6(a).

It is important to realize that what really distinguishes the tree-based algorithm and the degree-based algorithm is the *cluster maintenance overhead*. Since the degree of a node is a rather unstable parameter, in the degree-based algorithm, *every* link change (formation and breakage) has to be forwarded to *all* the cluster members. This is certainly not the case in the tree-based algorithm where, as long as the cluster is still diameter-2, link formation and link breakage are propagated in the HELLO beaconing message as described in Section 66.4 and will not be forwarded by the other nodes.

To take this point one step further, we count the total number of intra-cluster link changes during the simulation. We call a link change between nodes A and B in the same cluster *benign* if after the change nodes A and B remain in the original diameter-2 cluster, and A and B have a common two-hop neighbor. For example, in the cluster shown in Figure 66.5(a), the breakage of link (6,7) is benign since the resultant graph is still diameter-2, and nodes 6 and 7 have a common two-hop neighbor (node 8). However, the breakage of link (3,8) is not benign since nodes 3 and 8 do not have a common two-hop neighbor. We note that, trivially, the tree-based algorithm saves *at least* one message forwarding per benign link change over the degree-based algorithm. We count the number and ratio of benign link changes, and the corresponding simulation results are shown in Figure 66.8. As the simulation result shows, the ratio of benign link changes is quite significant, and as the node density becomes higher, the savings become more and more significant.

Our simulation results have revealed an interesting piece of evidence that speaks for the robustness of our tree-based algorithm: even when multiple link failures occur in a cluster, the probability of the existence of a maintenance leader is still very high. Theoretically, when multiple edges are removed from a diameter-2 graph, there may no longer exist a maintenance leader in the resulting graph. There are two approaches that can be employed by the tree-based algorithm to deal with this situation. The first approach is to predict link failure ahead of time whenever possible. Thus, when multiple link failures occur at the same time, all these links are actually still there, and the maintenance leader can arbitrarily choose one link as the only broken link. Essentially, this prevents real link failures from occurring in the first place. The second approach is to simply let multiple link failures occur. By Corollary 66.1, if a maintenance leader exists, each node will know the maintenance result in at most four message rounds. A node sets a 4-message round long timer when violation is detected. Upon time-out, each node uses the cluster initialization algorithm described in Subsection 66.4.4 as the last resort for cluster maintenance.

66.6 Application

In this section, we discuss *topology control* [17] in mobile ad hoc networks as a sample application of the cluster-based general-purpose infrastructure we have proposed. We propose a cluster-based algorithm to construct an approximate Minimum Spanning Tree (MST). The algorithm has three phases: Phase 1: cluster formation. A distributed clustering algorithm is used to generate and maintain clusters in the network. In this work, our focus is the diameter-2 clustering we have proposed. Phase 2: forming intra-cluster MST. In our infrastructure, since each cluster is diameter-2, a distributed MST algorithm exists that finishes very quickly [1]. Alternatively, a leader for topology control can be elected in each cluster, which is responsible for running a centralized MST algorithm (such as Kruskal's algorithm [10]). Note that this leader is a *logical* leader for the topology control application only. Phase 3: connecting clusters. In this phase, connectivity between adjacent clusters is considered. Each cluster runs the following algorithm: by exchanging information with neighboring clusters only, a cluster knows its *shortest* link to each of its adjacent clusters, as well as the shortest links between each pair of its adjacent clusters. Based on this information, a cluster constructs a Localized Minimum Spanning Tree (LMST) [18]. Note that each node in the LMST is a cluster, and each edge is the LMST is an actual link between two nodes. When running the LMST to establish connections between two adjacent clusters, the power assigned to the involved nodes is increased only. The collections of all edges in the LMSTs constructed by all nodes, as well as the links selected in Phase 2, form the resulting structure.

We have conducted a simulation study to determine the effectiveness of our cluster-based MST algorithm. In this study, 100–500 nodes were distributed uniformly at random in an area of $1000 * 1000\,\text{m}^2$. When operating at full transmission power, each node has a transmission range of 250 m. In the simulation, for a specific number of nodes, we generate 50 different topologies. And the result is the average of these 50 simulation runs. Also, in this simulation, we consider static topology only.

We consider the following metrics in the simulation: (1) The two most important metrics, average link length and number of links in the resulting topology, consider only the bidirectional links in the resulting connected structures. For a connected network with N nodes, its MST has $N - 1$ links. The average link length is calculated as the sum of the length of each link divided by the number of links. (2) The degree of the node is counted in the following way: for a node u with transmission power P_u, and a node v with transmission power P_v, if the distance between u and v is not larger than P_v, then node v is considered as a neighbor of u. Note that this relationship is not symmetric. (3) Average node power is calculated as the sum of the powers assigned to each node divided by the total number of nodes in the network. (4) Max node power: it is the maximum value among the powers assigned to the nodes in the network.

For any given topology, the diameter-2 clustering scheme can potentially generate/maintain fewer (or equal) number of clusters than any central-node-based clustering scheme, hence there are more topology information available for making *intra*-cluster decisions (since there are *more* nodes in a cluster) and for making *inter*-cluster decisions (since there are *fewer* clusters in the network), leading to a better-quality global structure.

Detailed simulation results are shown in Table 66.1. In the table, *MST* is the result using a centralized Kruskal's algorithm, *Diameter-2* and *LowestID* are the results of diameter-2 clustering and the lowestID clustering, respectively. From the simulation result, it is evident that resulting topology constructed by our cluster-based MST algorithm approximates the MST effectively in terms of all the four performance metrics used. Specifically, (1) the average link length of the resulting structure is very close to the optimal value (the approximation ratio is 1.06, 1.06, 1.05, 1.08, 1.05 as the number of nodes increases from 100 to 500); the number of links in the resulting structure is about three more than the optimal value, regardless of the number of nodes in the networks (the approximation ratio is 1.03, 1.02, 1.01, 1.01, 1.01 as the number of nodes increases from 100 to 500). (2) The average node degree keeps stable when the number of nodes increases (the approximation ratio is 1.15, 1.16, 1.14, 1.12, 1.16 as the number of nodes increases from 100 to 500). (3) The average node power is very close to the optimal value (as the number of nodes increases from 100 to 500; the approximation ratio of the average node power is 1.09, 1.08, 1.07, 1.07, 1.07).

TABLE 66.1 Performance Comparison of the Three Topology Control Algorithms

Number of Nodes	Algorithm	MST	Diameter-2	LowestID
100	Max node power	164.44	190.32	192.23
100	Average node power	82.19	89.89	90.89
100	Average node degree	2.51	2.89	2.93
100	Average link length	68.06	72.14	72.77
100	Number of links	99	102.42	103.20
200	Max node power	116.74	147.80	149.71
200	Average node power	57.73	62.51	62.88
200	Average node degree	2.51	2.90	2.92
200	Average link length	47.42	50.32	50.50
200	Number of links	199	202.58	202.94
300	Max node power	99.44	124.19	125.79
300	Average node power	46.97	50.41	50.53
300	Average node degree	2.50	2.85	2.86
300	Average link length	38.66	40.70	40.79
300	Number of links	299	302.78	303.06
400	Max node power	86.70	110.51	113.82
400	Average node power	40.28	42.94	43.15
400	Average node degree	2.51	2.81	2.83
400	Average link length	33.19	35.74	34.85
400	Number of links	399	402.92	403.52
500	Max node power	78.44	99.75	100.42
500	Average node power	36.00	38.36	38.48
500	Average node degree	2.51	2.80	2.81
500	Average link length	29.67	31.09	31.15
500	Number of links	499	503.58	504.04

(4) The approximation ratio of the max node power is a little high (1.16, 1.27, 1.25, 1.27, 1.27 as the number of nodes increases from 100 to 500). This is expected since max node power is determined by the critical part of a network. In fact, it is proved in Ref. [17] that it is impossible for any localized algorithm to construct a connected structure such that the max node power based on this structure is within a constant factor of that based on MST.

In the simulation result, the diameter-2 clustering consistently generates better-quality structures than the lowestID clustering in terms of all the performance metrics used; however the difference between the two is small. The reason is that the simulation is conducted on static topologies only, and under static topologies, the difference between these two clustering schemes is not as dramatic as the difference in face of mobility (see Figure 66.6(a)). The advantage of diameter-2 clustering scheme is expected to be more obvious in face of node mobility.

Finally, it is worth emphasizing that in this section we use MST construction as an illustration of the application of our proposed general-purpose infrastructure; but in fact, the cluster-based infrastructure provides a powerful general framework, and similar approaches can be used to establish many other global structures such as *strongly connected* graphs [19].

66.7 Concluding Remarks

A large number of clustering schemes for MANET have been proposed in the recent literature. In general, we believe that a clustering scheme that can generate a more *stable* and *symmetric* virtual infrastructure is especially suitable for MANET, and such a virtual infrastructure can be leveraged by a number of MANET applications without introducing traffic bottlenecks and single points of failure. To illustrate the feasibility of this concept we have proposed a tree-based cluster initialization/maintenance algorithm for MANET based on a number of properties of diameter-2 graphs. The resulting algorithm is cluster-centric and works in the presence of node mobility. Simulation results demonstrated the effectiveness of our algorithm when compared to other clustering schemes in the literature.

The tree-based clustering algorithm proposed in this chapter can be further generalized to achieve (*d1*, *d2*)-clustering in which: two clusters merge when the diameter of the resulting cluster is not larger than *d1*, and a cluster is split into several diameter-*d1* clusters if its diameter is larger than *d2*. By adaptively changing the values of *d1* and *d2*, a stable and symmetric general-purpose virtual infrastructure can be achieved efficiently in large-scale MANET.

References

[1] Lynch, N., *Distributed Algorithms*, Morgan Kaufmann Publishers, San Francisco, California, 1996.

[2] Banerjee, S. and Khuller, S., A clustering scheme for hierarchical control in multi-hop wireless networks, in *Proc. INFOCOM*, 2001.

[3] Krishna, P., Chatterjee, M., Vaidya, N., and Pradhan, D., A cluster-based approach for routing in ad-hoc networks, *ACM SIGCOMM Comput. Comm. Review*, 27(2), 49, 1997.

[4] Baker, D. J. and Ephremides, A., The architectural organization of a mobile radio network via a distributed algorithm, *IEEE Trans. Comm.*, 29(11), 1694, 1981.

[5] Basagni, S., Distributed clustering for ad hoc networks, *Proc. Intl. Symp. on Parallel Architectures, Algorithms, and Networks*, 1999, p. 310.

[6] Gerla, M. and Tsai, J., Multicluster, mobile, multimedia radio network, *Wireless Networks*, 1(3), 255, 1995.

[7] Lin, C. R. and Gerla, M., Adaptive clustering for mobile wireless networks, *J. Sel. Areas Comm.*, 15(7), 1265, 1997.

[8] Nakano, K. and Olariu, S., Randomized leader election protocols in ad-hoc networks, *Proc. Int. Symp. on Structural Inf. and Comm. Complexity*, 2000, p. 253.

[9] Wang, L., Ph.D. Thesis, Old Dominion University, 2005.

[10] Cormen, T. H., Leiserson, C. E., and Rivest, R. L., *Introduction to Algorithms*, MIT Press and McGraw-Hill, New York, 1992.

[11] Chiang, C. C., Wu, H. K., Liu, W., and Gerla, M., Routing in clustered multihop, mobile wireless networks with fading channel, *Proc. IEEE Singapore Int. Conf. on Networks*, MIT Press, Cambridge Massachussetts, 1997, p. 197.

[12] Huang, H., Richa, A. W., and Segal, M., Approximation algorithms for the mobile piercing set problem with applications to clustering in ad-hoc networks, *Proc. Int. Workshop on Disc. Algorithms and Methods for Mobile Comput. and Comm.*, 2002, p. 62.

[13] Amis, A. D., Prakash, R., Huynh, D., and Vuong, T., Max–min d-cluster formation in wireless ad hoc networks, *Proc. INFOCOM*, 1, 2000, p. 32.

[14] Camp, T., Boleng, J., and Davies, V., A survey of mobility models for ad hoc network research, *Wireless Comm. Mobile Comput.*, 2(5), 483, 2002.

[15] McDonald, A. B. and Znati, T., A mobility based framework for adaptive clustering in wireless ad-hoc networks, *IEEE J. Sel. Areas Comm. (Special Issue on Ad-Hoc Networks)*, 17(8), 1466, 1999.

[16] Ramanathan, S. and Steenstrup, M., Hierarchically-organized, multihop mobile networks for multimedia support, *Mobile Networks Appl.*, 3(1), 101, 1998.

[17] Li, X.-Y., Topology control in wireless ad hoc networks, in *Ad Hoc Networking*, Basagni, S., Conti, M., Giordano, S., and Stojmenovic, I., eds., John Wiley and Sons, Hoboken, New Jersey, 2003.

[18] Li, N., Hou, J., and Sha, L., Design and analysis of an MST-based topology control algorithm, *Proc. INFOCOM*, 2003.

[19] Shen, C., Srisathapornphat, C., Liu, R., Huang, Z., Jaikaeo, C., and Lloyd, E., CLTC: a cluster-based topology control framework for ad hoc networks, *IEEE Trans. Mobile Comput.*, 3(1), 1, 2004.

[20] Steenstrup, M., Cluster-based networks, *Ad Hoc Networking*, Addison-Wesley, Reading, MA, 2001.

67

Topology Control Problems for Wireless Ad Hoc Networks

Errol L. Lloyd[*]
University of Delaware

S. S. Ravi[†]
University at Albany—State University of New York

67.1 Introduction

An ad hoc network consists of a collection of transceivers (nodes). All communication among these nodes is based on radio propagation. The term "ad hoc" is used to indicate that the nodes organize themselves into a network without any preexisting infrastructure. Wireless ad hoc networks are used in a number of applications including military operations, cellular phones, and emergency search-and-rescue operations [1,2]. In such networks, battery power is a precious resource. This chapter presents approximation algorithms for several problems that arise in the context of wireless ad hoc networks, where the main goal is to minimize the power used by the nodes.

To develop precise formulations of the problems considered in this chapter, we need to introduce some basic concepts and terminology regarding the nodes used in the ad hoc network. For each ordered pair (u, v) of nodes, there is a **transmission power threshold**, denoted by $p(u, v)$, with the following significance: A signal transmitted by the node u can be received by v only when the transmission power of u is at least $p(u, v)$. The transmission power threshold for a pair of nodes depends on a number of factors including the distance between the nodes, the antenna gain at the sender and the receiver, interference, and noise [3].

Each assignment of powers to the nodes of an ad hoc network induces a directed graph in the following manner. The nodes of this directed graph are in one-to-one correspondence with the nodes of the ad hoc network. A directed edge (u, v) is in this graph if and only if the transmission power of u is at least

[*] E. L. Lloyd prepared through collaborative participation in the Communications and Networks Consortium sponsored by the U. S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. The U. S. Government is authorized to reproduce and distribute reprints for government purposes not withstanding any copyright notation thereon.

[†] S. S. Ravi was supported by NSF Grant CCR-97-34936.

the transmission power threshold $p(u, v)$. The main goal of **topology control** is to assign transmission powers to nodes so that the resulting directed graph satisfies some specified properties. Since the battery power of each node is an expensive resource, it is important to achieve the goal while minimizing a given function of the transmission powers assigned to the nodes. Examples of desirable graph properties are connectivity, small diameter, and so on. The primary minimization objectives considered in the literature are the maximum power assigned to a node and the total power assigned to all nodes (the latter objective is equivalent to minimizing the average power assigned to a node). Most of the results presented in this chapter deal with the minimum total power objective.

Unless otherwise mentioned, the power threshold values are assumed to be *symmetric*; that is, for any pair of nodes u and v, $p(u, v) = p(v, u)$. Under this assumption, the power threshold values can be represented by an undirected complete graph which we call the **threshold graph**. The weight of each edge $\{u, v\}$ in this graph is the transmission power threshold $p(u, v)$.

As stated above, the main motivation for studying topology control problems is to make efficient use of available power at each node. In addition, using a minimum amount of power at each node to achieve a given task is also likely to decrease the interference in the medium access (MAC) layer between adjacent radios. We refer the reader to Refs. [3–5] for a thorough discussion of the power control issues in ad hoc networks. References [6,7] present surveys on topology control. The emphasis of Ref. [6] is on how topology control can facilitate the design of routing protocols for ad hoc networks. Reference [7] provides a thorough coverage of the practical aspects of topology control.

Precise formulations of the optimization problems considered in this chapter are provided in Section 67.2. Many of these problems are **NP**-complete. The practical importance of these problems motivates the study of approximation algorithms for them. The focus of this chapter is on approximation algorithms with proven performance guarantees. Our goal is to discuss some known techniques for developing such approximation algorithms rather than to provide a survey on the topic of topology control.

The remainder of this chapter is organized as follows. Section 67.2 provides the necessary definitions and develops a notation for specifying topology control problems. Section 67.3 presents approximation algorithms for topology control problems where the goal is to minimize the total power assigned to the nodes of the network. Both centralized and distributed approximation algorithms are discussed in that section. Section 67.4 summarizes known approximation results for other objectives. Finally, some directions for future research are presented in Section 67.5.

67.2 Model and Problem Formulation

67.2.1 Graph Models for Topology Control

Topology control problems have been studied under two graph models. The discussion in Section 67.1 corresponds to the **directed graph model** studied in Ref. [3]. In the **undirected graph model** considered in Ref. [8], each power assignment induces an undirected graph in the following manner. An undirected edge $\{u, v\}$ is in this graph if and only if the transmission powers of u and v are both at least the transmission power threshold $p(u, v)$. Under this model, the goal of a topology control problem is to assign transmission powers to nodes such that the resulting undirected graph has a specified property, and a specified function of the powers assigned to nodes is minimized. Note that the directed graph model allows two-way communication between some pairs of nodes and one-way communication between other pairs of nodes. In contrast, every edge in the undirected graph model corresponds to a two-way communication.

67.2.2 Notation for Topology Control Problems

In general, a topology control problem can be specified by a triple of the form $\langle \mathbb{M}, \mathbb{P}, \mathbb{O} \rangle$. In such a specification, $\mathbb{M} \in \{\text{Dir}, \text{Undir}\}$ represents the graph model, $\mathbb{P}$ the desired graph property and $\mathbb{O}$ the minimization objective. For the problems considered in this chapter, $\mathbb{O} \in \{\text{MaxP}, \text{TotalP}\}$ (abbreviations of Max Power and Total Power). For example, in the ⟨Dir, Strongly Connected, MaxP⟩ problem, powers must be assigned to nodes so that the resulting directed graph (induced by the assigned powers in relation to the edges of the threshold graph) is strongly connected and the maximum power assigned to a node is minimized. Similarly, the ⟨Undir, 2-Node-Connected, TotalP⟩ problem seeks to assign powers to the nodes so that the resulting undirected graph has a node connectivity (see below for definition) of (at least) 2 and the sum of the powers assigned to all nodes is minimized. Throughout this chapter, an instance of an $\langle \mathbb{M}, \mathbb{P}, \mathbb{O} \rangle$ problem is specified by a threshold graph.

67.2.3 Additional Definitions

This section collects together the definitions of some graph theoretic and algorithmic terms used throughout this chapter.

Given an undirected graph $G(V, E)$, an **edge subgraph** $G'(V, E')$ of G has all of the nodes of G and the edge set E' is a subset of E. Further, if G is an edge-weighted graph, then the weight of each edge in G' is the same as it is in G.

The **node connectivity** of an undirected graph is the smallest number of nodes that must be deleted from the graph so that the resulting graph is disconnected. The **edge connectivity** of an undirected graph is the smallest number of edges that must be deleted from the graph so that the resulting graph is disconnected. For example, a tree has node and edge connectivities equal to 1 while a simple cycle has node and edge connectivities equal to 2. When the node (edge) connectivity of a graph is greater than or equal to k, the graph is said to be k-**node connected** (k-**edge connected**). Given an undirected graph, polynomial algorithms are known for finding its node and edge connectivities [9].

Some of the results presented in this chapter use the following definition.

Definition 67.1

A property $\mathbb{P}$ *of the (directed or undirected) graph induced by a power assignment is* monotone *if the property continues to hold even when the powers assigned to some nodes are increased while the powers assigned to the other nodes remain unchanged.*

Example 67.1

For any $k \geq 1$, the property k-Node-Connected for undirected graphs is monotone since increasing the powers of some nodes while keeping the powers of other nodes unchanged may only add edges to the graph. However, properties such as Acyclic or Bipartite are not monotone.

We say that an approximation algorithm provides a **performance guarantee** of ρ if for every instance of the problem, the solution produced by the approximation algorithm is within the multiplicative factor of ρ of the optimal solution.

67.3 Approximation Algorithms for Minimizing Total Power

67.3.1 A General Framework

Topology control problems in which the minimization objective is total power tend to be computationally intractable. For example, the problem is **NP**-hard even for the (simple) property 1-Node-Connected [8]. In this section, we first discuss a general framework developed in Ref. [10] for approximation algorithms for topology control problems of the form ⟨Undir, $\mathbb{P}$, TotalP⟩. We then discuss how approximation algorithms for several graph properties can be derived from this framework.

Input: An instance I of $\langle$UNDIR, $\mathbb{P}$, TOTALP$\rangle$ where the property $\mathbb{P}$ is monotone and polynomial-time testable.

Output: A power value $\pi(u)$ for each node u such that the undirected graph induced by the power assignment satisfies property $\mathbb{P}$ and the total power assigned to all nodes is as small as possible.

Steps:

1. Let $G_c(V, E_c)$ denote the threshold graph for the given problem instance.
2. Construct an edge subgraph $G'(V, E')$ of G_c such that G' satisfies property $\mathbb{P}$ and the total weight of the edges in E' is minimum among all edge subgraphs of G_c satisfying property $\mathbb{P}$.
3. For each node u, assign a power value $\pi(u)$ equal to the weight of the largest edge in E' incident on u.

FIGURE 67.1 Heuristic GEN-TOTAL-POWER: A general framework for approximating total power.

The framework assumes that the property $\mathbb{P}$ to be satisfied by the graph is *monotone* and that it can be tested in polynomial time. It also assumes *symmetric* power thresholds as in Refs. [8,11,12]; that is, for any pair of nodes u and v, the power thresholds $p(u, v)$ and $p(v, u)$ are equal.

The general approximation framework (called Heuristic GEN-TOTAL-POWER) is shown in Figure 67.1. Note that steps 1 and 3 of the heuristic can be implemented in polynomial time. The time complexity of step 2 depends crucially on the property $\mathbb{P}$. For some properties such as 1-NODE CONNECTED, step 2 can be done in polynomial time. For other properties such as 2-NODE CONNECTED, step 2 cannot be done in polynomial time, unless **P** = **NP** [13]. In such cases, an efficient algorithm that produces an approximately minimum solution can be used in step 2. Theorem 67.1 proves the correctness of the general approach and establishes its performance guarantee as a function of some parameters that depend on the property $\mathbb{P}$ and the approximation algorithm used in step 2 of the framework.

Theorem 67.1

Let I be an instance of $\langle$UNDIR, $\mathbb{P}$, TOTALP$\rangle$, *where $\mathbb{P}$ is a monotone property. Let $OPT(I)$ and $GTP(I)$ denote respectively the total power assigned to the nodes in an optimal solution and in a solution produced by Heuristic* GEN-TOTAL-POWER *for the instance I. Then, the following hold.*

(i) The graph G_π induced by the power assignment produced by the heuristic (i.e., step 3) satisfies property $\mathbb{P}$.

(ii) Let $H(V, E_H)$ be an edge subgraph of the threshold graph G_c such that H has the minimum total edge weight among all edge subgraphs of G_c satisfying property $\mathbb{P}$. Let $W(H)$ denote the total edge weight of H. Let step 2 of the heuristic produce an edge subgraph $G'(V, E')$ of G with total edge weight $W(G')$. Suppose there are quantities $\alpha > 0$ and $\beta > 0$ such that (a) $W(H) \leq \alpha\, OPT(I)$ and (b) $W(G') \leq \beta\, W(H)$. Then, $GTP(I) \leq 2\alpha\beta\, OPT(I)$. That is, Heuristic GEN-TOTAL-POWER *provides a performance guarantee of $2\alpha\beta$.*

Proof

Part (i). The edge subgraph $G'(V, E')$ constructed in step 2 of the heuristic satisfies property $\mathbb{P}$. We show that every edge in E' is also in the subgraph G_π induced by the power assignment π produced in step 3. Then, even if G_π has other edges, the monotonicity of $\mathbb{P}$ allows us to conclude that G_π satisfies $\mathbb{P}$.

Consider an edge $\{u, v\}$ with weight $p(u, v)$ in E'. Recall that $p(u, v)$ is the minimum power threshold for the existence of edge $\{u, v\}$ and that the power thresholds are symmetric. Since step 3 assigns to each node the maximum of the weights of edges incident on that node, we have $\pi(u) \geq p(u, v)$ and $\pi(v) \geq p(u, v)$. Therefore, the graph G_π induced by the power assignment also contains the edge $\{u, v\}$ and this completes the proof of Part (i).

Part (ii). By conditions (a) and (b) in the statement of the theorem, we have $W(G') \leq \alpha\, \beta\, OPT(I)$. We observe that $GTP(I) \leq 2\, W(G')$. This is because step 3 of the heuristic may assign the weight of any edge to at most two nodes (namely, the endpoints of the edge). Combining the two inequalities, we get $GTP(I) \leq 2\alpha\beta\, OPT(I)$. ■

67.3.2 Applications of Theorem 67.1

This section presents several examples of approximation algorithms derived from the general framework outlined in Figure 67.1.

67.3.2.1 An Approximation Algorithm for ⟨UNDIR, 1-NODE-CONNECTED, TOTALP⟩

As a first example, we observe that the 2-approximation algorithm presented in Ref. [8] for the ⟨UNDIR, 1-NODE-CONNECTED, TOTALP⟩ problem can be derived from the above general framework. In step 2 of the framework, the algorithm in Ref. [8] constructs a minimum spanning tree of G_c. It is also shown that the total power assigned by any optimal solution is at least the weight of a minimum spanning tree of G_c (see Lemma 67.9). Thus, using the notation of Theorem 67.1, $\alpha = \beta = 1$ for their approximation algorithm. Since 1-NODE-CONNECTED is a monotone property, it follows from Theorem 67.1 that the performance guarantee of their algorithm is 2. A different approximation algorithm with a performance guarantee of 2 for ⟨UNDIR, 1-NODE-CONNECTED, TOTALP⟩ was presented in Ref. [14]. The best known approximation algorithm for this problem provides a performance guarantee of $5/3 + \epsilon$, for any fixed $\epsilon > 0$ [15].

67.3.2.2 An Approximation Algorithm for ⟨UNDIR, 2-NODE-CONNECTED, TOTALP⟩

The approximation algorithm discussed in this section is from Ref. [10]. The **NP**-hardness of the ⟨UNDIR, 2-NODE-CONNECTED, TOTALP⟩ problem was established in Ref. [16]. We note that the property 2-NODE-CONNECTED is monotone. The following notation is used throughout this section. I denotes the given instance of ⟨UNDIR, 2-NODE-CONNECTED, TOTALP⟩ with n nodes. For each node $u, \pi^*(u)$ denotes the power assigned to u in an optimal solution. Further, $OPT(I)$ denotes the sum of the powers assigned to the nodes in an optimal solution.

The approximation algorithm in Ref. [10] for the ⟨UNDIR, 2-NODE-CONNECTED, TOTALP⟩ problem is obtained from the framework of Figure 67.1 by using an approximation algorithm from Ref. [17] for the minimum-weight 2-NODE-CONNECTED subgraph problem in step 2. This approximation algorithm provides a performance guarantee of $(2 + 1/n)$. Thus, using the notation of Theorem 67.1, we have $\beta \leq (2 + 1/n)$.

Lemma 67.1 below shows that the threshold graph $G_c(V, E_c)$ of the instance I contains an edge subgraph $G_1(V, E_1)$ such that G_1 is 2-NODE-CONNECTED and the total weight $W(G_1)$ of the edges in G_1 is at most $(2 - 2/n)\, OPT(I)$. Again, using the notation of Theorem 67.1, this result implies that $\alpha \leq (2 - 2/n)$.

Thus, once Lemma 67.1 is established, it would follow from Theorem 67.1 that the performance guarantee of the resulting approximation algorithm is $2\,(2 - 2/n)\,(2 + 1/n)$, which approaches 8 asymptotically from below. The remainder of this section is devoted to the formal statement and proof of Lemma 67.1.

Lemma 67.1

Let I denote an instance of the ⟨UNDIR, 2-NODE-CONNECTED, TOTALP⟩ *problem with n nodes. Let $G_c(V, E_c)$ denote the threshold graph for the instance I. Let $OPT(I)$ denote the total power assigned to the nodes in an optimal solution to I. There is an edge subgraph $G_1(V, E_1)$ of G_c such that G_1 is* 2-NODE-CONNECTED *and the total weight $W(G_1)$ of the edges in G_1 is at most* $(2 - 2/n)\, OPT(I)$.

Our proof of Lemma 67.1 begins with an optimal power assignment π^* to instance I and constructs a graph G_1 satisfying the properties mentioned in the above statement. This construction relies on several definitions and known results from graph theory.

Definition 67.2

Let $G(V, E)$ be an undirected graph. Suppose the node sequence $\langle v_1, v_2, v_3, \ldots, v_k, v_1\rangle$ forms a simple cycle C of length at least 4 in G. Any edge $\{v_i, v_j\}$ of G $(1 \leq i \neq j \leq k)$ which is not in C is a chord.

Definition 67.3

An undirected graph $G(V, E)$ is critically 2-NODE-CONNECTED *if it satisfies both of the following conditions: (i) G is* 2-NODE-CONNECTED. *(ii) For every edge $e \in E$, the subgraph of G obtained by deleting the edge e is* not 2-NODE-CONNECTED.

For example, a simple cycle on three or more nodes is critically 2-NODE-CONNECTED. This is because such a cycle is 2-NODE-CONNECTED, and deleting any edge of the cycle yields a simple path which is not 2-NODE-CONNECTED. A number of properties of critically 2-NODE-CONNECTED graphs have been established in the literature (see, e. g., Refs. [18–20]). In proving Lemma 67.1. we use the following property established[1] in Refs. [18,19].

Proposition 67.1

If a graph G is critically 2-NODE-CONNECTED *then no cycle of G has a chord.* ■

We also need some terminology associated with **Depth-First-Search** (DFS) [21]. When DFS is carried out on a connected undirected graph $G(V, E)$, a spanning tree $T(V, E_T)$ is produced. Each edge in T is called a **tree edge**. Each tree edge joins a child to its parent. An **ancestor** of a node u in T is a node which is not the parent of u but which is encountered in the path from u to the root of T. Each edge in $E - E_T$ is called a **back edge**. Each back edge joins a node u to an ancestor of u in T. The following lemma establishes a simple property of back edges that arise when DFS is carried out on a critically 2-NODE-CONNECTED graph.

Lemma 67.2

Let $G(V, E)$ be a critically 2-NODE-CONNECTED *graph and let $T(V, E_T)$ be a spanning tree for G produced using DFS. For any node u, there is at most one back edge from u to an ancestor of u in T.*

Proof

The proof is by contradiction. Suppose a node u has two or more back edges. Let v and w be two ancestors of u in T such that both $\{u, v\}$ and $\{u, w\}$ are back edges. Note that these two edges are in G. Without loss of generality, let w be encountered before v in the path in T from the root to u. The path from w to u in T together with the edge $\{u, w\}$ forms a cycle in G. By our choice of w, this cycle also includes the node v. Therefore, the edge $\{u, v\}$ is a chord in the cycle. This contradicts the assumption that G is critically 2-NODE-CONNECTED since by Proposition 67.1, no cycle in G can have a chord. The lemma follows. ■

We now prove several additional lemmas that are used in our proof of Lemma 67.1. Consider the given instance I of the $\langle$UNDIR, 2-NODE-CONNECTED, TOTALP$\rangle$ problem and let V denote the set of nodes. For the chosen optimal power assignment π^*, let p^* denote the maximum power value assigned to a node. Let the chosen optimal power assignment induce the graph G_{π^*}. Note that G_{π^*} is 2-NODE-CONNECTED. Let $G_1^*(V, E_1^*)$ be an edge subgraph of G_{π^*} such that G_1^* is critically 2-NODE-CONNECTED. Such a subgraph can be obtained by starting with G_{π^*} and repeatedly removing edges until no further edge deletion

[1] It should be noted that the graph theoretic terminology used in Refs. [18,19] is different from that used in this chapter. The statement of Proposition 67.1 given above is from Ref. [20].

is possible without violating the property. For each edge $\{u, v\}$ of G_1^*, we assign a weight $w_1(u, v)$ as follows:

1. Let r be a node such that $\pi^*(r) = p^*$. Using r as the root, perform a DFS of G_1^*. Let $T(V, E_T)$ be the resulting spanning tree. Thus, each edge of G_1^* is either a tree edge or a back edge.
2. For each tree edge $\{u, v\}$ where v is the parent of u, let $w_1(u, v) = \pi^*(u)$.
3. For each back edge $\{u, v\}$ where v is an ancestor of u, let $w_1(u, v) = \pi^*(u)$.

We can now bound the total weight $W_1(G_1^*)$ of the edges in G_1^* under the edge weight function w_1.

Lemma 67.3

$W_1(G_1^*) \le (2 - 2/n)\, OPT(I)$.

Proof

As mentioned above, each edge of G_1^* is either a tree edge or a back edge. Consider the tree edges first. For each tree edge $\{u, v\}$, where v is the parent of u, $w_1(u, v) = \pi^*(u)$. Thus, the weight $\pi^*(u)$ is assigned to at most one tree edge (namely, the edge that joins u to the parent of u if any in T). The power value of the root r in the optimal solution, namely p^*, is not assigned to any tree edge (since the root has no parent). Thus, the total weight of all of the tree edges under the weight function w_1 is bounded by $OPT(I) - p^*$.

Now consider the back edges. For each back edge $\{u, v\}$, where v is an ancestor of u, $w_1(u, v) = \pi^*(u)$. Since G_1^* is critically 2-NODE-CONNECTED, by Lemma 67.2, each node has at most one back edge to an ancestor. Thus, the weight $\pi^*(u)$ is assigned to at most one back edge. Again, the power value p^* of the root r in the optimal solution is not assigned to any back edge. Thus, the total weight of all of the back edges under the weight function w_1 is also bounded by $OPT(I) - p^*$.

Therefore, the total weight $W_1(G_1^*)$ of all of the edges in G_1^* under the edge-weight function w_1 is at most $2\, OPT(I) - 2p^*$. Since p^* is the largest power value assigned to a node in the optimal solution, p^* is at least $OPT(I)/n$. Hence, $W_1(G_1^*)$ is bounded by $(2 - 2/n)\, OPT(I)$ as required. ■

The following lemma relates the weight $w_1(u, v)$ of an edge $\{u, v\}$ to the power threshold $p(u, v)$ needed for the existence of the edge.

Lemma 67.4

For any edge $\{u, v\}$ *in* G_1^*, $p(u, v) \le w_1(u, v)$.

Proof

Consider any edge $\{u, v\}$ in G_1^*. Since G_1^* is an edge subgraph of G_{π^*} (the graph induced by the chosen optimal power assignment), $\{u, v\}$ is also an edge in G_{π^*}. Also, recall that the minimum power threshold values are symmetric. Therefore, $\pi^*(u) \ge p(u, v)$ and $\pi^*(v) \ge p(u, v)$. Hence, $\min\{\pi^*(u), \pi^*(v)\} \ge p(u, v)$. The weight assigned to the edge $\{u, v\}$ by the edge-weight function w_1 is either $\pi^*(u)$ or $\pi^*(v)$. Therefore, $w_1(u, v) \ge \min\{\pi^*(u), \pi^*(v)\}$. It follows that $w_1(u, v) \ge p(u, v)$. ■

We are now ready to complete the proof of Lemma 67.1.

Proof of Lemma 67.1

Starting from the optimal power assignment π^*, construct the graph $G_1^*(V, E_1^*)$ as described above. Since the threshold graph G_c is complete, every edge in G_1^* is also in G_c. Consider the edge subgraph $G_1(V, E_1)$ of G_c where $E_1 = E_1^*$. Since G_1^* is 2-NODE-CONNECTED, so is G_1. By Lemma 67.4, for each edge $\{u, v\}$ in E_1, $p(u, v) \le w_1(u, v)$. Therefore, the total weight $W(G_1)$ of all of the edges in G_1 under the edge-weight function p is at most $W_1(G_1^*)$. By Lemma 67.3, $W_1(G_1^*)$ is bounded by $(2 - 2/n)\, OPT(I)$. Therefore, $W(G_1)$ is also bounded by $(2 - 2/n)\, OPT(I)$. In other words, the edge subgraph $G_1(V, E_1)$ is 2-NODE-CONNECTED and the total weight of all its edges is at most $(2 - 2/n)\, OPT(I)$. This completes the proof of Lemma 67.1. ■

Theorem 67.2 is a direct consequence of the above discussion.

Theorem 67.2

There is a polynomial-time approximation algorithm with a performance guarantee of $2\,(2-2/n)\,(2+1/n)$ *(which approaches* 8 *asymptotically from below) for the* ⟨UNDIR, 2-NODE-CONNECTED, TOTALP⟩ *problem.* ■

67.3.2.3 An Approximation Algorithm for ⟨UNDIR, 2-EDGE-CONNECTED, TOTALP⟩

A result analogous to Theorem 67.2 has also been obtained in Ref. [10] for the ⟨UNDIR, 2-EDGE-CONNECTED, TOTALP⟩ problem, where the goal is to induce a graph that is 2-EDGE-CONNECTED. This problem has also been shown to be NP-complete in Ref. [16]. To obtain an approximation algorithm for this problem from the general framework, an approximation algorithm of Khuller and Vishkin [22] is used. Their approximation algorithm produces a 2-EDGE-CONNECTED subgraph whose cost is at most twice that of a minimum 2-EDGE-CONNECTED subgraph. In the notation of Theorem 67.1, we have $\beta \leq 2$. Again using the notation of Theorem 67.1, it is also possible to show that $\alpha \leq (2-1/n)$. The proof of this result is almost identical to that for the 2-NODE-CONNECTED case, except that we need an analog of Proposition 67.1. Before stating this analog, we have the following definition (which is analogous to Definition 67.3).

Definition 67.4

An undirected graph $G(V, E)$ is critically 2-EDGE-CONNECTED *if it satisfies both of the following conditions: (i) G is* 2-EDGE-CONNECTED. *(ii) For every edge $e \in E$, the subgraph of G obtained by deleting the edge e is* not 2-EDGE-CONNECTED.

The following is the analog of Proposition 67.1 for critically 2-EDGE-CONNECTED graphs.

Proposition 67.2

If a graph G is critically 2-EDGE-CONNECTED *then no cycle of G has a chord.*

Proof

The proof is by contradiction. Suppose G is critically 2-EDGE-CONNECTED but there is a cycle $C = \langle v_1, v_2, \ldots, v_r\rangle$, with $r \geq 4$, with a chord $\{v_i, v_j\}$. Consider the graph G' obtained from G by deleting the chord $\{v_i, v_j\}$. We will show that G' is 2-EDGE-CONNECTED, thus contradicting the assumption that G is critically 2-EDGE-CONNECTED.

To show that G' is 2-EDGE-CONNECTED, it suffices to show that G' cannot be disconnected by deleting any single edge. Consider any edge $\{x, y\}$ of G', and let G'' denote the graph created by deleting $\{x, y\}$ from G'. Since we deleted only one edge from G', all the nodes of the cycle C are in the same connected component of G''. Thus, if we create the graph G_1 by adding the chord $\{v_i, v_j\}$ to G'', the two graphs G_1 and G'' have the same number of connected components. However, G_1 is also the graph obtained by deleting the edge $\{x, y\}$ from G. Since G is 2-EDGE-CONNECTED, G_1 is connected. Thus, G'' is also connected. We therefore conclude that G' is 2-EDGE-CONNECTED, and this contradiction completes the proof of Proposition 67.2. ■

The remainder of the proof to show that $\alpha \leq (2-2/n)$ is identical to that for the 2-NODE-CONNECTED case. With $\alpha \leq (2-2/n)$ and $\beta \leq 2$, Theorem 67.3 is a direct consequence of Theorem 67.1.

Theorem 67.3

There is a polynomial-time approximation algorithm with a performance guarantee of $8(1-1/n)$ *(which approaches* 8 *asymptotically from below) for the* ⟨UNDIR, 2-EDGE-CONNECTED, TOTALP⟩ *problem.* ■

67.3.2.4 Improvement by Calinescu and Wan

The approximation algorithms of Sections 67.3.2.2 and 67.3.2.3 were shown to provide a performance guarantee of at most 8 using the general bound given in Theorem 67.1. By a more careful analysis of the two algorithms, Calinescu and Wan [16] have shown that the algorithms actually provide a performance guarantee of at most 4. In particular, they show that the approximation algorithm of Section 67.3.2.3 for the ⟨UNDIR, 2-EDGE-CONNECTED, TOTALP⟩ problem can be generalized to obtain an approximation algorithm with a performance guarantee of $2k$ for the ⟨UNDIR, k-EDGE-CONNECTED, TOTALP⟩ problem, for any $k \geq 2$. Here, we discuss this general result.

We need to introduce some notation. Given an undirected graph G, the directed graph obtained by replacing each undirected edge $\{u, v\}$ by the pair of directed edges (u, v) and (v, u) is denoted by $\overrightarrow{G}$. When each edge $e = \{u, v\}$ of the graph G has an edge weight $w(e)$, the weights of the directed edges (u, v) and (v, u) in $\overrightarrow{G}$ are also set to $w(e)$. For a directed graph $D(V, A)$, we use $\overline{D}$ to denote the undirected graph obtained from D by erasing the directions on all the edges and combining multiedges into a single edge. For an edge-weighted undirected graph G (directed graph D), the total weight of all the edges is denoted by $W(G)$ $(W(D))$.

Suppose $G(V, E)$ is the undirected graph induced by assigning powers to the nodes of an ad hoc network. For any node $u \in V$, the **power of** u **with respect to** G, denoted by $p_G(u)$, is given by $p_G(u) = \max\{p(u, v) : \{u, v\} \in E\}$. The **power for inducing** G, denoted by $P(G)$, is given by $P(G) = \sum_{u \in V} p_G(u)$.

We will also need the directed graph model (Section 67.1) of graphs induced by power assignments. Recall that in that model, a directed edge (u, v) is in the induced directed graph if and only if the power assigned to u is at least $p(u, v)$. The definitions of power of a node and the power for inducing a graph given above for undirected graphs can be readily extended to directed graphs. For any node u of a directed graph $D(V, A)$, $p_D(u)$ is given by $p_D(u) = \max\{p(u, v) \;:\; (u, v) \in A\}$. Likewise, $P(D)$ is given by $P(D) = \sum_{u \in V} p_D(u)$.

A directed graph is **strongly** k**-edge-connected** if for each pair of nodes u and v, there are at least k edge-disjoint paths from u to v. A directed graph $D(V, A)$ is an **inward branching** rooted at a node $s \in V$ if $|A| = |V| - 1$ and there is a directed path from each node in $V - \{s\}$ to s. Thus, in an inward branching rooted at s, the outdegree of each node except s is 1 and the outdegree of s is 0. The following three-part observation is a simple consequence of the above definitions.

Observation 67.1

Let G be an undirected graph and D be a directed graph, both having edge weights. Then, the following hold: (i) $P(G) \leq 2W(G)$. (ii) $W(\overline{D}) \leq W(D)$. (iii) If D is an inward branching, then $P(D) = W(D)$.

Given a directed graph $G(V, A)$, a vertex $s \in V$ and an integer $k \geq 1$, we say that D is k**-Edge-Inconnected** to s if there are at least k edge-disjoint paths from each vertex in $V - \{s\}$ to s. The following known results about k-EDGE-INCONNECTED graphs are used in proving the main result of this section.

Theorem 67.4

(a) *Suppose $D(V, A)$ is* k-EDGE-INCONNECTED *to a vertex $s \in V$. Then A contains k pairwise disjoint subsets $A_1, A_2, \ldots, A_k$ such that for each i, $1 \leq i \leq k$, the directed graph $B_i(V, A_i)$ is an inward branching rooted at s [23].*

(b) *Given a directed graph $D(V, A)$ with a nonnegative weight for each edge, an integer $k \geq 2$ and a vertex $s \in V$, the problem of obtaining a subgraph $D_1(V, A_1)$ of minimum total weight such that D_1 is* k-EDGE-INCONNECTED *to s can be solved in polynomial time [24,25].* ■

Recall that the approximation algorithm of Section 67.3.2.3 was obtained using the Khuller–Vishkin Algorithm (KV-Algorithm) [22] in step 2 of the general framework. Relying on part (b) of Theorem 67.4, the KV-Algorithm actually provides a performance guarantee of 2 for the following problem: Given an integer $k \geq 2$ and an undirected and k-EDGE-CONNECTED graph G with nonnegative edge weights, find a subgraph $H(V, E_H)$ of minimum weight such that H is also k-EDGE-CONNECTED. We give below the steps of the KV-Algorithm since they are used in the proof of the main result.

1. From the graph G, construct $\overrightarrow{G}$.
2. Choose any vertex s of G and construct a subgraph D of $\overrightarrow{G}$ such that D is k-EDGE-INCONNECTED to s and has the smallest total weight among all such subgraphs. (By Part (b) of Theorem 67.4, this step can be done in polynomial time.)
3. Construct and output $\overline{D}$.

It is shown in Ref. [22] that $\overline{D}$ is k-EDGE-CONNECTED and that $W(D)$ is at most twice the weight of an optimal solution.

We are now ready to prove the main result of this section. The proof below follows the presentation in Ref. [16].

Theorem 67.5

For any $k \geq 2$, the approximation algorithm obtained by using the KV-Algorithm in step 2 of the general framework (Figure 67.1) provides a performance guarantee of $2k$ *for the* ⟨UNDIR, k-EDGE-CONNECTED, TOTALP⟩ *problem.*

Proof

Let I denote the given instance of the ⟨UNDIR, k-EDGE-CONNECTED, TOTALP⟩ problem. Let $OPT(I)$ and $HEU(I)$ denote respectively the total power of an optimal assignment π^* and that of the power assignment π produced by the heuristic referred to in the statement of the theorem. Note that π^* is an optimal power assignment under the undirected graph model. Our goal is to show that $HEU(I) \leq 2k\, OPT(I)$.

Consider an optimal power assignment π_d^* for the instance I under the *directed* graph model. Let π_d^* induce the directed graph $D^*(V, A^*)$. Note that D^* is STRONGLY k-EDGE-CONNECTED. Since the power threshold values are symmetric, it follows that $P(D^*) \leq OPT(I)$.

Consider any vertex $s \in V$. Since $D^*(V, A^*)$ is STRONGLY k-EDGE-CONNECTED, D^* is also k-EDGE-INCONNECTED to s. By Part (a) of Theorem 67.4, A^* contains k pairwise disjoint subsets $A_1, A_2, \ldots, A_k$ such that each directed graph $B_i(V, A_i)$, $1 \leq i \leq k$, is an inward branching with s as the root. Thus, the directed graph $D'(V, \cup_{i=1}^{k} A_i)$ is k-EDGE-INCONNECTED to s. Therefore, the weight of the solution D found in step 2 of the KV-Algorithm is at most $W(D')$. Using this observation, the following claim provides a bound on $W(D)$.

Claim 67.1

$W(D) \leq k\, OPT(I)$.

Proof

As mentioned above, $W(D) \leq W(D')$. Since $W(D') = \sum_{i=1}^{k} W(B_i)$, we have

$$\begin{aligned}
W(D) &\leq \sum_{i=1}^{k} W(B_i) \\
&= \sum_{i=1}^{k} P(B_i) \quad \text{(Part (iii) of Observation 67.1)} \\
&\leq \sum_{i=1}^{k} P(D^*) \\
&= k\, P(D^*) \\
&\leq k\, OPT(I) \quad (\text{since}\, p(D^*) \leq OPT(I))
\end{aligned}$$

as indicated in the statement of the claim.

The power assignment π output by the approximation algorithm is obtained from the graph $\overline{D}$. Therefore,

$$\begin{aligned} HEU(I) &= P(\overline{D}) \\ &\le 2\,W(\overline{D}) && \text{(Part (i) of Observation 67.1)} \\ &\le 2\,W(D) && \text{(Part (ii) of Observation 67.1)} \\ &\le 2k\,OPT(I) && \text{(Claim 67.1).} \end{aligned}$$

This completes the proof of Theorem 67.5. ■

67.3.2.5 Bicriteria Approximation for Diameter and Total Power

We now present an application of Theorem 67.1 to the topology control problem where the goal is to induce a graph whose diameter is bounded by a given value. For an undirected graph $G(V, E)$, recall that the **diameter** is given by $\max\{d(u, v) \;:\; u, v \in V\}$, where $d(u, v)$ denotes the number of edges in a shortest path between nodes u and v. Graphs with small diameters are desirable in the context of ad hoc networks since the diameter determines the largest end-to-end delay in such a network. We assume that a diameter bound γ is given as part of the problem instance and the goal is to find a power assignment such that the diameter of the induced graph is at most γ and the total power assigned to the nodes is minimized. For this problem, denoted by ⟨UNDIR, DIAMETER, TOTALP⟩, approximation algorithms with similar performance guarantees were presented in Refs. [26,27]. We discuss the algorithm from Ref. [26].

The ⟨UNDIR, DIAMETER, TOTALP⟩ problem involves two minimization objectives, namely, the diameter and the total power. Such **bicriteria** minimization problems are typically handled by designating one of the objectives as a budget constraint and the other as the minimization objective [28]. Following Ref. [26], we will designate the diameter bound as a constraint and total power as the minimization objective. It is shown in Ref. [26] that if the diameter constraint must be satisfied, the total power cannot be approximated to within a factor $\lambda \log n$, for some λ, $0 < \lambda < 1$, unless **P** = **NP**. So, the approximation algorithm presented in Ref. [26] is a **bicriteria approximation**, where the diameter constraint is violated by a certain factor and the total power is approximated to within another factor. A formal definition of bicriteria approximation is as follows.

Definition 67.5

Suppose a problem Π with two minimization objectives A and B is posed in the following manner: Given a budget constraint on objective A, find a solution which minimizes the value of objective B among all solutions satisfying the budget constraint. An (ρ_1, ρ_2)-approximation algorithm *for problem Π is a polynomial-time algorithm that provides for every instance of Π a solution satisfying the following two conditions.*

1. *The solution violates the budget constraint on objective A by a factor of at most ρ_1.*
2. *The value of objective B in the solution is within a factor of at most ρ_2 of the minimum possible value satisfying the budget constraint.*

To obtain bicriteria approximation algorithms for the ⟨UNDIR, DIAMETER, TOTALP⟩ problem, we rely on known approximation results for another problem, called the **Minimum Cost Tree with a Diameter Constraint** (MCTDC), also involving two minimization objectives. A formal definition of this problem is as follows.

Minimum Cost Tree with a Diameter Constraint

Instance: A connected undirected graph $G(V, E)$, a nonnegative weight $w(e)$ for each edge $e \in E$, an integer $\gamma \le n - 1$.

Requirement: Find a spanning tree $T(V, E_T)$ of G such that $\text{DIA}(T) \le \gamma$ and the total edge weight of T is minimum among all the trees satisfying the diameter constraint.

MCTDC is known to be NP-hard [28]. Bicriteria approximations for this problem have been presented in Refs. [28–30].

Input: An instance I of the $\langle$Undir, Diameter, TotalP$\rangle$ problem, with diameter bound γ.

Output: A power value $\pi(u)$ for each node u such that the diameter of the undirected graph induced by the power assignment is close to γ and the total power assigned to all nodes is as small as possible.

1. Let $G_c(V, E_c)$ denote the threshold graph for the given instance of the $\langle$Undir, Diameter, TotalP$\rangle$ problem.
2. Use any approximation algorithm $\mathcal{A}$ for the Mctdc problem on $G_c(V, E_c)$ with diameter bound 2γ, and obtain a spanning tree $T(V, E_T)$ of G_c.
3. For each node u, assign a power value $\pi(u)$ equal to the weight of the largest edge incident on u in T.

FIGURE 67.2 Outline of Heuristic Gen-Diameter-Total-Power.

The bicriteria approximation algorithm for $\langle$Undir, Diameter, TotalP$\rangle$ in Ref. [26] is based on the general framework (Figure 67.1). The steps of the approximation algorithm, which we call Gen-Diameter-Total-Power, are shown in Figure 67.2. In step 2 of Figure 67.2, any approximation algorithm $\mathcal{A}$ for the Mctdc problem can be used. As long as $\mathcal{A}$ runs in polynomial time, Gen-Diameter-Total-Power also runs in polynomial time. The performance guarantee provided by Gen-Diameter-Total-Power is a function of the performance guarantee provided by Algorithm $\mathcal{A}$.

The solution produced by Heuristic Gen-Diameter-Total-Power is approximate in terms of both diameter and total power. So, we cannot directly rely on Theorem 67.1 to derive the performance guarantee provided by the heuristic.

For the remainder of this section, we use the following notation. Let I denote the given instance of the $\langle$Undir, Diameter, TotalP$\rangle$ problem with n nodes and diameter bound $\gamma \geq 1$. Let π^* denote an optimal power assignment such that the graph G_{π^*} induced by π^* has diameter at most γ, and let $OPT(I) = \sum_{v \in V} \pi^*(v)$. Let π denote the power assignment produced by the heuristic and let G_π denote the graph induced by π. Let $DTP(I) = \sum_{v \in V} \pi(v)$, the total power assigned by the heuristic for the instance I. The bicriteria approximation result proved in this section is as follows.

Theorem 67.6

Suppose Algorithm $\mathcal{A}$ used in step 2 of Heuristic Gen-Diameter-Total-Power *is a (ρ_1, ρ_2)-approximation algorithm for the* Mctdc *problem. For any instance I of the* $\langle$Undir, Diameter, TotalP$\rangle$ *problem, Heuristic* Gen-Diameter-Total-Power *produces a power assignment π satisfying the following two properties: (1)* $\text{Dia}(G_\pi) \leq 2\,\rho_1\,\gamma$. *(2) $DTP(I) \leq 2\,\rho_2\,(1 - 1/n)\;OPT(I)$. In other words,* Gen-Diameter-Total-Power *provides a $(2\,\rho_1, 2\rho_2(1 - 1/n))$ bicriteria approximation for the* $\langle$Undir, Diameter, TotalP$\rangle$ *problem.*

Several lemmas are needed to prove Theorem 67.6. We begin with a simple lemma about spanning trees generated by carrying out a **breadth-first-search** (BFS) on a connected graph [21].

Lemma 67.5

Let G be a connected graph with diameter δ. Let T be any spanning tree for G generated by BFS. Then $\text{Dia}(T) \leq 2\delta$.

Proof

Suppose T was generated by carrying out BFS starting with node v (as the root). For any node x, the distance $d(v, x)$ between the root v and x in T is the length of a shortest path between them in G [21]. Thus, for any node x, $d(v, x) \leq \delta$ in T. It follows that the maximum distance between any pair of nodes in T, that is $\text{Dia}(T)$, is at most 2δ. ■

The next lemma indicates why in step 2 of Heuristic Gen-Diameter-Total-Power, the diameter bound of 2γ is used.

Lemma 67.6

Consider the threshold graph $G_c(V, E_c)$ of the ⟨UNDIR, DIAMETER, TOTALP⟩ *instance I. There is a spanning tree $T_1(V, E_{T_1})$ of G_c satisfying the following two properties. (a)* $\text{DIA}(T_1) \leq 2\gamma$. *(b) Let $W(E_{T_1})$ denote the total edge weight of T_1. Then, $W(E_{T_1}) \leq (1 - 1/n)\, OPT(I)$.*

Proof

Part (a). Consider the graph G_{π^*} induced by the optimal power assignment π^*. Note that $\text{DIA}(G_{\pi^*}) \leq \gamma$. Let v be a node such that $\pi^*(v)$ has the largest value among all the nodes in V. Let $T_1(V, E_{T_1})$ be a spanning tree of G_{π^*} generated by carrying out a BFS on G_{π^*} with v as the root. Then, from Lemma 67.5, we have $\text{DIA}(T_1) \leq 2\gamma$.

Part (b). Consider another assignment w of weights to the edges of T_1 as indicated below. Consider each edge $\{x, y\}$ in T_1, where y is the parent of x. Let $w(x, y) = \pi^*(x)$. Thus, the power value assigned by the optimal solution to each node except the root becomes the weight of exactly one edge of T_1. The power value $\pi^*(v)$ of the root is not assigned to any edge. Therefore,

$$\sum_{\{x,y\}\in E_{T_1}} w(x, y) = OPT(I) - \pi^*(v)$$

Since v has the maximum power value under π^* among all the nodes, we have $\pi^*(v) \geq OPT(I)/n$. Therefore,

$$\sum_{\{x,y\}\in E_{T_1}} w(x, y) \leq (1 - 1/n)\, OPT(I)$$

The following claim relates the weight $w(x, y)$ to the power threshold value $p(x, y)$.

Claim 67.2

For each edge $\{x, y\} \in E_{T_1}$, $w(x, y) \geq p(x, y)$.

Proof of Claim

Consider edge $\{x, y\} \in E_{T_1}$, where y is the parent of x. Then, by definition, $w(x, y) = \pi^*(x)$. Since T_1 is a spanning tree of G_{π^*}, the edge $\{x, y\}$ is also in G_{π^*}. This fact, in conjunction with the assumption that the power threshold values are symmetric, implies that $\pi^*(x) \geq p(x, y)$. The claim follows.

As a simple consequence of the above claim, we have

$$W(E_{T_1}) \leq \sum_{\{x,y\}\in E_{T_1}} w(x, y) \leq (1 - 1/n)\, OPT(I)$$

and this completes the proof of Part (b) of the lemma. ■

The next lemma uses the performance guarantee provided by the approximation algorithm $\mathcal{A}$ used in step 2 of the heuristic.

Lemma 67.7

Let $T(V, E_T)$ denote the tree produced by $\mathcal{A}$ at the end of step 2 of Heuristic GEN-DIAMETER-TOTAL-POWER. *Let $W(E_T)$ denote the total weight of the edges in T. Let (ρ_1, ρ_2) denote the performance guarantee provided by $\mathcal{A}$ for the* MCTDC *problem. Then the following hold: (a)* $\text{DIA}(T) \leq 2\,\rho_1\,\gamma$. *(b) $W(E_T) \leq \rho_2\,(1 - 1/n)\, OPT(I)$.*

Proof

By Lemma 67.6, the edge-weighted graph $G_c(V, E_c)$ has a spanning tree of diameter at most 2γ and total edge weight at most $(1 - 1/n)\, OPT(I)$. Now, Lemma 67.7 is a simple consequence of the performance guarantee provided by Algorithm $\mathcal{A}$. ■

We are now ready to prove Theorem 67.6.

Proof of Theorem 67.6

Consider the spanning tree $T(V, E_T)$ produced in step 2 of the heuristic. It is straightforward to verify that every edge $\{x, y\} \in E_T$ is also in $G_\pi(V, E_\pi)$, the graph induced by the power assignment constructed in step 3 of the heuristic. Since $\text{DIA}(T) \leq 2\,\rho_1\,\gamma$, and the addition of edges cannot increase the diameter, it follows that $\text{DIA}(G_\pi) \leq 2\,\rho_1\,\gamma$.

To bound *DTP*(*I*), we note from Lemma 67.7 that $W(E_T) \leq \rho_2\,(1 - 1/n)\,OPT(I)$. In the power assignment constructed in step 3, the weight of any edge can be assigned to at most two nodes (namely, the end points of that edge). Thus, the total power assigned to all the nodes is at most $2W(E_T)$. In other words, $DTP(I) \leq 2\rho_2\,(1 - 1/n)\,OPT(I)$. ■

We now briefly indicate how several bicriteria approximation algorithms for the ⟨UNDIR, DIAMETER, TOTALP⟩ problem can be obtained using GEN-DIAMETER-TOTAL-POWER in conjunction with known bicriteria approximation results for the MCTDC problem.

1. For any $\epsilon > 0$, a $(2\,\lceil \log_2 n \rceil, (1+\epsilon)\,\lceil \log_2 n \rceil)$-approximation algorithm is presented in Ref. [28] for the MCTDC problem. Using this algorithm and setting $\epsilon < 1/n$, we can obtain a $(4\,\lceil \log_2 n \rceil, 2\,\lceil \log_2 n \rceil)$-approximation algorithm for the ⟨UNDIR, DIAMETER, TOTALP⟩ problem.
2. For any *fixed* $\gamma \geq 1$, a $(1, O(\gamma \log n))$-approximation algorithm for the MCTDC problem is presented in Ref. [30]. Thus, for any fixed $\gamma \geq 1$, we can obtain a $(2, O(\gamma \log n))$-approximation algorithm for the ⟨UNDIR, DIAMETER, TOTALP⟩ problem.
3. For any γ and any fixed $\epsilon > 0$, a $(1, O(n^\epsilon \log n))$-approximation algorithm for the MCTDC problem is presented in Ref. [29]. Thus, for this case, we can obtain a $(2, O(n^\epsilon \log n))$-approximation algorithm for the ⟨UNDIR, DIAMETER, TOTALP⟩ problem.

67.3.3 Some Extensions

The general framework for minimizing total power can also be used to obtain polynomial-time approximation algorithms for topology control problems wherein the connectivity requirements are specified using *proper functions* [31]. To obtain this result, the general method outlined in Refs. [31,32] is used as the algorithm in step 2 of the framework. The method in Refs. [31,32] gives a 2-approximation algorithm for network design problems specified using proper functions. Using the notation of Theorem 67.1, $\beta = 2$. It is also straightforward to show that the threshold graph contains an appropriate subgraph of weight at most the optimal solution value. In other words, $\alpha \leq 1$. Thus, we obtain a 4-approximation algorithm for the general class of problems defined in Refs. [31,32]. An important example of a problem in this class is the Steiner variant of connectivity, where the goal is to assign power levels so as to induce a graph which connects a specified subset of nodes, called **terminals** (possibly using nodes which are not terminals). An approximation algorithm with a performance guarantee of $(1 + \ln\sqrt{3})$ is known for the Steiner tree problem in graphs [33]. Thus, using this approximation algorithm, the approach yields a $(2 + \ln 3)$-approximation for the Steiner variant.

The bicriteria results for minimizing the diameter and total power can also be extended to the Steiner version, where only the terminals need to be connected together into a graph of bounded diameter. Letting η denote the number of terminals, Ref. [28] presents an $(O(\log \eta), O(\log \eta))$-approximation algorithm for the Steiner version of the MCTDC problem. Using this approximation algorithm in step 2 of Figure 67.2, we obtain an $(O(\log \eta), O(\log \eta))$-approximation algorithm for the Steiner version of the ⟨UNDIR, DIAMETER, TOTALP⟩ problem.

67.3.4 Distributed Approximation Algorithms

Previous sections considered centralized approximation algorithms for minimizing total power used by the nodes. In this section we discuss *distributed* topology control algorithms for minimizing total power. These algorithms also have provably good performance guarantees.

67.3.4.1 Preliminaries

The distributed algorithms that we discuss all utilize the *geometric model.* Typically, in geometric instances of topology control problems, nodes are points in a metric space, and the transmission power threshold $p(u, v)$ is determined by the distance $d(u, v)$ between u and v. Specifically, the power threshold $p(u, v)$ is $d(u, v)^\alpha$, where α is the *attenuation constant* associated with path loss. The *path loss* is the ratio of the received power to the transmitted power of the signal [34]. The value of α is typically between 2 and 4.

For consistency with the notation used in previous sections, we assume that instances of topology control problems under the geometric model are also specified through threshold graphs, where the threshold values are determined from the distances as discussed above. The following result providing a *weak triangle inequality* between the threshold values under the geometric model is established in Ref. [35].

Lemma 67.8

For nodes u, v, and w, $p(u, v) \leq 2^{\alpha-1}(p(u, w) + p(w, v))$. ■

It is shown in Ref. [16] that:

Theorem 67.7

The ⟨Geometric, 2-Node-Connected, Total Power⟩ *problem is* ***NP****-hard.* ■

67.3.4.2 Distributed Algorithms for ⟨Geometric, 2-Node-Connected, TotalP⟩

A framework for distributed topology control algorithms for the ⟨Geometric, 2-Node-Connected, TotalP⟩ problem is given in Figure 67.3. The framework consists of first finding a minimum spanning tree of the threshold graph, and then, for each nonleaf node u in that tree, adding edges to connect all of the neighbors of u.

The framework given in Figure 67.3 is easy to implement in a distributed fashion: step 1 utilizes the distributed minimum spanning tree algorithm given by Gallager et al. [36]; the computation in step 2 is handled by each node u, which then distributes the relevant results to its neighbors; and, each node computes its own power value in step 3.

Theorem 67.8

The framework in Figure 67.3 produces a 2-Node-Connected *network.*

Proof

Since the resulting network contains a minimum spanning tree of G_t, the network is at least 1-Node-Connected. Thus, by way of contradiction, assume that G' is not 2-Node-Connected. This means that

Input: A threshold graph G_t under the geometric model.

Output: A power value $\pi(u)$ for each node u such that the undirected graph induced by that power assignment is 2-Node-Connected.

Steps:

1. Compute a minimum spanning tree T of G_t and let $G' = T$.
2. For each nonleaf node u in T, let V_u consist of the neighbors of u in T. Then, from G_t, add edges E_u to G' such that the graph $G_u = (V_u, E_u)$ is connected.
3. For each node u, assign a power value $\pi(u)$ equal to the largest threshold among the edges incident on u in G'.

FIGURE 67.3 A distributed framework for ⟨Geometric, 2-Node-Connected, Total Power⟩.

some node u is an articulation point of G', hence u must be a node of degree two or more in the minimum spanning tree T that was placed into G'. Let v and w be neighbors of u that are in different connected components if u is removed from G'. This is not possible since step 2 of the framework adds edges to G' to connect all of the neighbors of u without using any of the edges in T. ■

While the framework does produce a network that is 2-NODE-CONNECTED, the specific method used to connect the neighbors of node u in step 2 is left open. Two natural methods for connecting the neighbors of u are ones in which:

- Each graph G_u is a simple path [35].
- Each graph G_u is a minimum spanning tree of the restriction of G_t to V_u [16].

The quality of the solutions produced by algorithms based on the framework in Figure 67.3 is dependent both on the path loss exponent (α) and on the particular method utilized in step 2 to connect the neighbors of node u. To state this dependence, we use the following notation. For any instance I of ⟨GEOMETRIC, 2-NODE-CONNECTED, TOTAL POWER⟩, let $OPT(I)$ and $DF(I)$ denote the total power assigned by an optimal solution and that produced by the above framework, respectively. We begin with the following result which was originally shown in Ref. [35]:

Theorem 67.9

Let I be an instance of ⟨GEOMETRIC, 2-NODE-CONNECTED, TOTAL POWER⟩. *Using the distributed framework and finding a simple path in step 2, $DF(I) \leq (2 + 2^{\alpha+2})\ OPT(I)$.*

To prove this theorem we begin by letting $C(G)$ denote the sum of the threshold values associated with the edges in a graph G. Note that this is different from the total power value needed to induce G. We have the following lemma whose proof also appears in Ref. [8].

Lemma 67.9

For G_t, let π' be an optimal power assignment such that the induced graph $G_{\pi'}$ is 1-NODE-CONNECTED. *Then $C(T) \leq \sum_{v \in G_t} \pi'(v)$, where T is the minimum spanning tree computed in step 1 of the framework.*

Proof

Consider any spanning tree T' of $G_{\pi'}$, and any leaf x in T'. The threshold value of the edge between x and its parent y, is no more than the power value of x in $G_{\pi'}$. Now remove node x and edge $\{x, y\}$ from the tree. By recursively applying this node removal, it is seen that the weight of each edge in T' is bounded from above by the power value assigned to some node, and each node is utilized at most once in this fashion. Thus, $C(T') \leq \sum_{v \in G_t} \pi'(v)$. The lemma follows since T' is also a spanning tree of G_t, and T is a minimum spanning tree of G_t. Hence $C(T) \leq C(T')$. ■

Proof of Theorem 67.9

Consider the graph G' computed by the framework. Since each edge in G' determines the power assignment to at most two nodes, it follows that

$$DF(I) \ \leq\ 2C(G') \tag{67.1}$$

Note that G' consists of the minimum spanning tree T computed in step 1 along with the edges added in step 2. Let G'' be a graph over the nodes of G' that contains all of the edges added in step 2. Since the edges of T and G'' are disjoint,

$$C(G') \ =\ C(T) + C(G'') \tag{67.2}$$

Consider any edge $\{u, v\}$ in G''. Recall that u and v are both neighbors of some w, such that edges $\{w, u\}$ and $\{w, v\}$ are both in T. From Lemma 67.8, $p(u, v) \leq 2^{\alpha-1}(p(u, w) + p(w, v))$. Note that for any edge $\{s, t\}$ in T, there are at most four edges of G'' that are adjacent to it. Thus,

$$C(G'') \ \leq\ 4 * 2^{\alpha-1}C(T) \ =\ 2^{\alpha+1}C(T) \tag{67.3}$$

Combining Eqs. (67.1), (67.2), and (67.3), we have

$$DF(I) \le 2(C(T) + 2^{\alpha+1}C(T)) = (2 + 2^{\alpha+2})C(T)$$

By applying Lemma 67.9, it follows that $DF(I) \le (2 + 2^{\alpha+2}) \sum_{v \in G_t} \pi'(v)$. Finally, the theorem follows by noting that $\sum_{v \in G_t} \pi'(v)$ cannot exceed the optimal power $OPT(I)$ needed to produce a 2-NODE-CONNECTED network. ■

Recall from above that finding a minimum spanning tree in step 2 is an alternative to finding a simple path. For that case a somewhat lower approximation ratio was shown in Ref. [16].

Theorem 67.10

Let I be an instance of ⟨GEOMETRIC, 2-NODE-CONNECTED, TOTAL POWER⟩. *Using the distributed framework and finding a minimum spanning tree in step 2, $DF(I) \le 8\ OPT(I)$ when $\alpha = 2$ and $DF(I) \le (3.2 * 2^{\alpha})$ $OPT(I)$ for any $\alpha > 2$.* ■

The above discussion focused on distributed algorithms for ⟨GEOMETRIC, 2-NODE-CONNECTED, TOTAL POWER⟩. Using that framework, one can also obtain the following result for ⟨GEOMETRIC, 1-NODE-CONNECTED, TOTAL POWER⟩.

Theorem 67.11

Let I be an instance of ⟨GEOMETRIC, 1-NODE-CONNECTED, TOTAL POWER⟩, *using a distributed algorithm consisting of steps 1 and 3 of the framework, $DF(I) \le 2\ OPT(I)$.* ■

That is, the algorithm[2] finds a minimum spanning tree of G_t, and the result follows from the proof of Theorem 67.9.

67.4 A Summary of Results for Other Topology Control Problems

The literature contains approximation results for a variety of topology control problems. In this section we provide a brief overview of some of these results.

Results similar to those presented in Section 67.3.2 have been obtained for higher node connectivities [37]. Two of the results presented in that paper are the following. For any $k \ge 3$, they present a $3k$-approximation algorithm for the problem of inducing a k-NODE-CONNECTED graph. For the special range $3 \le k \le 7$, they also present an approximation algorithm with a performance guarantee of $k+2\lceil (k+1)/2 \rceil$.

Ref. [26] considers the topology control problem where the goal is to compute a power assignment π such that the undirected graph G_π is connected, the degree of each node in G_π is at least a specified value Δ, and the total power is minimized. It is shown there that the problem is **NP**-complete for every fixed integer $\Delta \ge 2$. Also, an approximation algorithm based on the general framework in Figure 67.1 with a performance guarantee of $2(\Delta + 1)(1 - 1/n)$ is presented for the problem.

Topology control problems where the power threshold values may be *asymmetric* were considered in Refs. [26,38]. In this model, there may be pairs of nodes u and v such that $p(u, v) \ne p(v, u)$. It was shown in Refs. [26,38] that the ⟨UNDIR, 1-NODE-CONNECTED, TOTALP⟩ problem under this model can be approximated to within $O(\log n)$ and that this result cannot be improved beyond constant a factor, unless **P** = **NP**. This is in sharp contrast to the symmetric threshold case, where there is a $(5/3+\epsilon)$-approximation algorithm (Section 67.3.2.1).

As mentioned in our initial formulation of topology control problems (Section 67.2.2), the objective of minimizing the maximum power assigned to any node has also been studied [3,10]. For any monotone and efficiently testable property $\mathbb{P}$, it was shown in Ref. [10] that there is a polynomial-time algorithm for

[2] In fact, this is the algorithm given in Ref. [8] for the ⟨UNDIR, 1-NODE-CONNECTED, TOTALP⟩ problem.

minimizing the maximum power. This algorithm makes $O(\log n)$ calls to an algorithm for testing whether a graph G has the property $\mathbb{P}$. It was also shown in Ref. [10] that for any given $\epsilon > 0$, it is possible to get a $(1 + \epsilon)$-approximation for minimizing the maximum power using only $O(\log\log(p_{max}/p_{min}))$ calls to the algorithm for testing $\mathbb{P}$, where p_{max} and p_{min} are the largest and the smallest power thresholds respectively. This approximation algorithm is particularly useful when the ratio p_{max}/p_{min} is small.

The polynomial algorithm for minimizing maximum power given in Ref. [10] may assign the maximum power value to a larger than necessary number of nodes. The problem of minimizing the number of nodes to which the maximum power is assigned was considered in Ref. [39]. It was shown that even for simple properties such as 1-NODE-CONNECTED, the problem is **NP**-complete. Further, it was shown that for any property, the problem can be reduced in an approximation-preserving manner to the problem of minimizing the total power. As a consequence, the $(5/3 + \epsilon)$-approximation algorithm for ⟨UNDIR, 1-NODE-CONNECTED, TOTALP⟩ given in Ref. [15] provides the same performance guarantee for minimizing the number of nodes assigned the maximum power. However, as pointed out in Ref. [15], the algorithm is not practical since it involves solving large linear programs. In Ref. [39], a 5/3-approximation algorithm was presented for the problem. This algorithm does not involve solving linear programs, and its running time is $O(n^3\alpha(n))$, where $\alpha(n)$ is the functional inverse of the Ackermann function [21].

Other topology control problems which address issues such as power assignment using game theoretic considerations and maximizing network lifetime are considered in Refs. [38,40,41].

67.5 Directions for Future Research

We start by mentioning some open problems in the context of minimizing total power. First, it is of interest to investigate whether there are approximation algorithms with better performance guarantees for inducing graphs with various node and edge connectivity requirements. A second open problem is whether there is a better bicriteria approximation algorithm for the problem of inducing a graph whose diameter is bounded by a specified value. A more general research issue is to identify other graph topologies that are useful in the context of ad hoc networks and to study the approximation issues for inducing such graph topologies.

Most of the results mentioned in this chapter are for symmetric power thresholds. Investigating approximation algorithms for inducing various graph topologies under the asymmetric power thresholds is also of interest. All of the known theoretical results on topology control are for ad hoc networks in which nodes of the network are stationary. Extending these results to the case where the nodes are mobile is a challenging research direction. In practice, centralized algorithms are likely to be of limited value. So, the development of distributed approximation algorithms for inducing graphs with various topologies is necessary if such algorithms are to be deployed in actual networks.

Disclaimer

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U. S. Government.

Acknowledgments

We thank Professor Madhav V. Marathe (VBI and Virginia Tech) and Dr. Ram Ramanathan (BBN) for their valuable suggestions.

References

[1] Perkins, C. E., Ed., *Ad Hoc Networking*, Addison-Wesley, Boston, MA, 2001.

[2] Stojmenovic, I., Ed., *Handbook of Wireless Networks and Mobile Computing*, Wiley, New York, NY, 2002.

[3] Ramanathan, R. and Rosales-Hain, R., Topology control of multihop wireless networks using transmit power adjustment, *Proc. IEEE INFOCOM*, 2000, p. 404.

[4] Royer, E. M., Melliar-Smith, P., and Moser L., An Analysis of the Optimum node density for ad hoc mobile networks, *Proc. IEEE Int. Conf. on Comm.*, 2001, p. 857.

[5] Radoplu, V. and Meng, T. H., Minimum energy mobile wireless networks, *IEEE J. Sel. Areas Comm.*, 17(8), 1333, 1999.

[6] Rajaraman, R., Topology control and routing in ad hoc networks: a survey, *SIGACT News*, 33(2), 60, 2002.

[7] Ramanathan, R., Antenna beam forming and power control for ad hoc networks, in *Mobile Ad Hoc Networking*, Basagni, S., Conti, M., Giordano, S., and Stojmenovic, I., Eds., IEEE Press/Wiley, New York, NY, 2004, p. 139.

[8] Kirousis, L. M., Kranakis, E., Krizanc, D., and Pelc, A., Power consumption in packet radio networks, *Theor. Comp. Sci.*, 243(1–2), 289, 2000.

[9] van Leeuwen, J., Graph Algorithms, in *Handbook of Theoretical Computer Science*, Vol. A, van Leeuwen, J., Ed., MIT Press and Elsevier, Cambridge, MA, 1990, chap. 10.

[10] Lloyd, E. L., Liu, R., Marathe, M. V., Ramanathan, R., and Ravi, S. S., Algorithmic aspects of topology control problems for ad hoc networks, *Mobile Networks and Appl.*, 10(1–2), 19, 2005.

[11] Clementi, A. E. F., Penna, P., and Silvestri, R., Hardness results for the power range assignment problem in packet radio networks, *Proc. Int. Workshop on Randomization and Approximation in Comp. Sci.*, Lecture Notes in Computer Science, Vol. 1671, Springer, Berlin, 1999, p. 195.

[12] Clementi, A. E. F., Penna, P., and Silvestri, R., The power range assignment problem in packet radio networks in the plane, *Proc. STACS*, 2000, p. 651.

[13] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Co., San Francisco, CA, 1979.

[14] Chen, W. and Huang, N., The strongly connecting problem on multihop packet radio networks, *IEEE Trans. Comm.*, 37(3), 293, 1989.

[15] Althaus, E., Calinescu, G., Mandoiu, I., Prasad, S., Tchervenski, N., and Zelikovsky, A., Power efficient range assignment for symmetric connectivity in static ad hoc wireless networks, *Wireless Networks*, 12(3), 287, 2006.

[16] Calinescu, G. and Wan, P.-J., Symmetric high connectivity with minimum total power consumption in multihop packet radio networks, *Proc. Int. Conf. on Ad hoc and Wireless Networks,* Lecture Notes in Computer Science, Vol. 2865, Springer, Berlin, 2003, p. 235.

[17] Khuller, S. and Raghavachari, B., Improved approximation algorithms for uniform connectivity problems, *J. Algorithms*, 21(2), 434, 1996.

[18] Dirac, G. A., Minimally 2-connected graphs, *J. Reine Angewandte Mathematik*, 228, 204, 1967.

[19] Plummer, M. D., On minimal blocks, *Trans. AMS*, 134, 85, 1968.

[20] West, D. B., *Introduction to Graph Theory*, 2nd ed., Prentice-Hall, Inc., Englewood Cliffs, NJ, 2001.

[21] Cormen, T., Leiserson, C., Rivest, R., and Stein, C., *Introduction to Algorithms*, 2nd ed., MIT Press and McGraw-Hill, Cambridge, MA, 2001.

[22] Khuller, S. and Vishkin, U., Biconnectivity approximations and graph carvings, *JACM*, 41(2), 214, 1994.

[23] Edmonds, J., Edge-disjoint branchings, in *Combinatorial Algorithms*, Rustin, R., Ed., Algorithmic Press, New York, NY, 1972, p. 91.

[24] Edmonds, J., Matroid Intersection, *Ann. Disc. Math.*, 4, 185, 1979.

[25] Gabow, H. N., A matroid approach to finding edge connectivity and packing arborescences, *Proc. STOC*, 1991, p. 112.

[26] Krumke, S. O., Liu, R., Lloyd, E. L., Marathe, M. V., Ramanathan, R., and Ravi, S. S., Topology control problems under symmetric and asymmetric power thresholds, *Proc. Int. Conf. on Ad hoc and Wireless Networks*, Lecture Notes in Computer Science, Vol. 2865, Springer, Berlin, 2003, p. 187.

[27] Calinescu, G., Kapoor, S., and Sarwat, M., Bounded-hops power assignment in ad-hoc wireless networks, *Proc. IEEE Wireless Comm. and Networking Conf.*, 2004, p. 1494 (complete version to appear in *Disc. Appl. Math.*).

[28] Marathe, M. V., Ravi, R., Sundaram, R., Ravi, S. S., Rosenkrantz, D. J., and Hunt, H. B., III, Bicriteria network design problems, *J. Algorithms*, 28(1), 142, 1998.

[29] Kortsarz, G. and Peleg, D., Approximating the weight of shallow light trees, *Disc. Appl. Math.*, 93(2–3), 265, 1999.

[30] Charikar, M., Chekuri, C., Cheung, T., Dai, Z., Goel, A., Guha, S., and Li, M., Approximation algorithms for directed Steiner problems, *J. Algorithms*, 33(1), 73, 1999.

[31] Goemans, M. and Williamson, D. P., A general approximation technique for constrained forest problems, *SIAM J. Comput.*, 24(2), 296, 1995.

[32] Agrawal, A., Klein, P., and Ravi, R., When trees collide: an approximation algorithm for the generalized Steiner problem on networks, *SIAM J. Comput.*, 24(3), 440, 1995.

[33] Robins, G. and Zelikovsky, A., Improved Steiner tree approximation in graphs, *Proc. SODA,* 2000, p. 770.

[34] Rappaport, T. S., *Wireless Communications: Principles and Practice*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1996.

[35] Hajiaghayi, M. T., Immorlica, N., and Mirrokni, V., Power optimization in fault-tolerant topology control algorithms for wireless multi-hop networks, *Proc. MobiCom*, 2003, p. 300.

[36] Gallager, R., Humblet, P., and Spira, P., A distributed algorithm for minimum-weight spanning trees, *ACM Trans. Prog. Lang. Syst.*, 5(1), 66, 1983.

[37] Jia, X., Kim, D., Makki, S., Wan, P., and Yi, C., Power assignment for *k*-connectivity in wireless ad hoc networks, *J. Comb. Opt.*, 9(2), 213, 2005.

[38] Calinescu, G., Kapoor, S., Olshevsky, A., and Zelikovsky, A., Network lifetime and power assignment in ad-hoc wireless networks, *Proc. European Symp. on Algorithms*, Lecture Notes in Computer Science, Vol. 2832, Springer, Berlin, 2003, p. 114.

[39] Lloyd, E. L., Liu, R., and Ravi, S. S., Approximating the minimum number of maximum power users in Ad-hoc networks, *Mobile Networks and Applications* (MONET), 11(2), 129, 2006.

[40] Kumar, V. S. A., Eidenbenz, S., and Zust, S., Equilibria in topology control games for ad hoc networks, *Proc. DIALM-POMC Workshop*, 2003, p. 2.

[41] Floreen, P., Kaski, P., Kohonen, J., and Orponen, P., Multicast time maximization in energy-constrained wireless networks, *IEEE J. Sel. Areas Comm.*, 23(1), 117, 2005.

68

Geometrical Spanner for Wireless Ad Hoc Networks

Xiang-Yang Li
Illinois Institute of Technology

Yu Wang
University of North Carolina at Charlotte

68.1 Introduction

Wireless ad hoc networks have been undergoing a revolution that promises to have a significant impact throughout society, one that could quite possibly dwarf milestones in the information revolution. Unlike traditional fixed infrastructure networks, there is no centralized control over ad hoc networks, which consist of an arbitrary distribution of radios in certain geographical area. Wireless ad hoc networks trigger many challenging research problems, as it intrinsically has many special characteristics and some unavoidable limitations, compared to other wired or wireless network. An important requirement of these networks is that they should be self-organizing, that is, transmission ranges and data paths are dynamically restructured with changing topology. Energy conservation and network performance are probably the most critical issues in wireless ad hoc networks, because wireless devices are usually powered by batteries only and have limited computing capability and memory. Recently, significant research [1–8] has been conducted on designing power-efficient network topology for ad hoc networks. Many proposed methods applied computational geometry technique (specifically, geometrical spanner) to achieve the power efficiency. In this chapter, we will review these approximation algorithms of power spanner for ad hoc networks.

68.1.1 Geometrical Spanner

Geometrical spanners have been studied intensively in computational geometry literature for years [9–15]. Let $G = (V, E)$ be a n-vertex connected weighted graph. The distance in G between two vertices $u, v \in V$ is the total weight (length) of the shortest path between u and v and is denoted by $d_G(u, v)$. A subgraph $H = (V, E')$, where $E' \subseteq E$, is a *t-spanner* of G if for every $u, v \in V$, $d_H(u, v) \leq t \cdot d_G(u, v)$ where t is a constant and called the *stretch factor*. In other words, if H is a t-spanner of G, for any path $\Pi(u, v)$ in G from a node $u \in V$ to another node $v \in V$ there is always a path $\Pi'(u, v)$ in H which has length at most t times of $\Pi(u, v)$. The spanner H can be treated as a constant (t) approximation of the original graph G in sense of shortest path length. If the spanner H is a sparse graph (i.e., the number of edges in H is linear with the number of vertices), it is called a *sparse spanner*.

68.1.2 Ad Hoc Networks: Graph Model

A wireless ad hoc network consists of a set V of n wireless nodes distributed in a two-dimensional plane. Each node has the same *maximum* transmission range R. By a proper scaling, we assume that all nodes have the maximum transmission range equal to one unit. These wireless nodes then define a *unit disk graph, UDG*(V), in which there is an edge between two nodes if and only if their Euclidean distance is at most one. In other words, we assume that two nodes can always receive the signal from each other directly if the Euclidean distance between them is no more than one unit. Hereafter, *UDG*(V) is always assumed to be connected. We also assume that all wireless nodes have distinctive identities and each wireless node knows its position information either through a low-power global position system (GPS) receiver or some other ways. By one-hop broadcasting, each node u can gather the location information of all nodes within its transmission range. In the most common power-attenuation model, the power to support a link uv is assumed to be $\|uv\|^{\beta}$, where $\|uv\|$ is the Euclidean distance between u and v, β is a real constant between 2 and 5 depending on the wireless transmission environment.

68.1.3 Topology Control: Power Spanner

The *topology control* techniques are to let each wireless device adjust its transmission range and select certain neighbors for communication, while maintaining a structure (network topology) that can support energy-efficient routing and improve the overall network performance. Remember that we use a unit disk graph (UDG) to model the original communication graph for an ad hoc network, and the UDG provides information about all possible topologies. Most topology control methods construct a sparse subgraph of UDG and then restrict the routings on the constructed subgraph to save energy. Not every connected subgraph of the UDG are suitable to be the network topology. A good network topology should be energy efficient, that is to say, the total power consumption of the shortest path (most power-efficient path) between any two nodes in final topology should not exceed a constant factor of the power consumption of the shortest path in original network. Given a path $v_1 v_2 \cdots v_h$ connecting two nodes v_1 and v_h, the energy cost of this path is $\sum_{j=1}^{h-1} \|v_j v_{j+1}\|^{\beta}$. The path with the least energy cost is called the shortest path in a graph. Borrowing the concept of spanner from computational geometry, we define, a subgraph H is called a *power spanner* of a graph G if there is a positive real constant $\rho_H(G)$ such that for any two nodes, the power consumption of the shortest path in H is at most $\rho_H(G)$ times of the power consumption of the shortest path in G. The constant $\rho_H(G)$ is called the *power stretch factor*. Similarly, we can define the length stretch factors $\ell_H(G)$ by setting $\beta = 1$. And we call a graph H a *length spanner* if the $\ell_H(G)$ is bounded by a constant. It is not difficult to show that, for any $H \subseteq G$ with a length stretch factor δ, its power stretch factor is at most δ^{β} for any graph G. In particular, a graph with a constant bounded length stretch factor must also have a constant bounded power stretch factor, but the reverse is not true. Finally, the power stretch factor has the following monotonic property: If $H_1 \subset H_2 \subset G$ then the power stretch factors of H_1 and H_2 satisfy $\rho_{H_1}(G) \geq \rho_{H_2}(G)$.

Besides the power efficiency (power spanners), the network topology should also have some of (or all of) the following desirable features: connected, sparse, planar, degree bounded, fault tolerant, etc.

68.1.4 Efficient Localized Construction

Unlike traditional wired network and cellular wireless networks, the wireless devices are often moving during the communication, which could change the network topology to some extent. Hence, it is more challenging to design a topology control algorithm for ad hoc networks: the topology should be locally and self-adaptively maintained without affecting the whole network, and the communication cost during maintaining should not be too high. In other words, the construction algorithm is preferred to be *localized.* Here, a distributed algorithm constructing a graph G is a *localized algorithm* if every node u can exactly decide all edges incident on u based only on the information of all nodes within a constant hops of u, that is, it runs in a constant number of rounds. More importantly, we expect that the total communication cost of the algorithm is $O(n)$ messages, where each message is $O(\log n)$ bits.

68.1.5 Organization

The rest of the chapter is organized as follows. In Section 68.2, we survey several planar spanners which are used as routing topologies for ad hoc networks. In Section 68.3, we survey several degree bounded spanners based on Yao graph. In Section 68.4, we present several methods to build degree bounded and planar spanners. In Section 68.5, more kinds of spanners are reviewed. We present our conclusions in Section 68.6.

68.2 Planar Spanner

Many geometric routing algorithms require the planar topology to guarantee the message delivery, such as right-hand routing, greedy face routing (GFG) [16], greedy perimeter stateless routing (GPSR) [17], adaptive face routing (AFR) [18], etc. Therefore, in this section, we review the methods to build planar spanners.

68.2.1 Relative Neighborhood Graph

Let $G = (V, E)$ be a geometric graph defined on vertex set V with edge set E. The *relative neighborhood graph*, denoted by RNG(G), is a geometric concept proposed by Toussaint [19,20]. It consists of all edges $uv \in E$ such that there is no point $w \in V$ with edges uw and wv in E satisfying $\|uw\| < \|uv\|$ and $\|wv\| < \|uv\|$. Thus, an edge uv is included if the intersection of two circles centered at u and v and with radius $\|uv\|$ do not contain any vertex w from the set V such that edges uw and wv exist in E. See Figure 68.1(a) for an illustration. When G is a UDG, we use RNG(V) to denote the graph instead of RNG(G). RNG(V) is a planar graph (i.e., no two edges cross each other), which also implies its sparseness: $|RNG(V)| \leq 3n$, where n is the number of vertices. For an undirected and connected graph G, RNG(G) is connected and contains the minimum spanning tree (MST) of G. In other words, if the UDG(V) is connected, the RNG(V) is connected too. This insures the connectivity of the ad hoc networks. Another important property is that RNG(V) can be constructed easily using a localized method. From the definition, each node only need information of its one-hop neighbors to construct the RNG(V). RNG(V) was used for efficient broadcasting (minimizing the number of retransmissions) in one-to-one broadcasting model in Ref. [21] and was used by the GPSR routing protocol [17] as routing topology that guarantees the delivery of the packet. However, the analysis by Bose et al. [11] implied that the length stretch factor of RNG(V) is

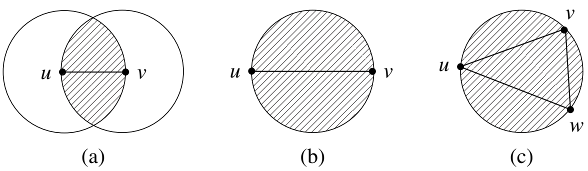


FIGURE 68.1 Illustration of definitions of planar structures: (a) RNG, (b) RNG, and (c) Del. The shaded areas ((a) the lune, (b) the diametric circle, and (c) the circumcircle of *uvw*) are empty of nodes inside.

at most $n-1$. Li et al. [3] and Wang [22] first studied and analyzed the power stretch factors of RNG(V), they showed that the power stretch factor of RNG(V) is actually $n-1$ by constructing an example. Thus, in summary, RNG(V) is not a power/length spanner, that is, RNG(V) is not power efficient for unicast routing in ad hoc networks.

68.2.2 Gabriel Graph

Let *disk*(u, v) be the disk with diameter uv. Then, the *Gabriel graph* [23] GG(G) contains an edge uv from G if and only if *disk*(u, v) contains no other vertex $w \in V$ such that there exist edges uw and wv from G. See Figure 68.1(b) for an illustration. When G is a UDG, we use GG(V) to denote the graph. GG(V) is also a popular planar graph. It is easy to show that RNG(V) is a subgraph of GG(V). Thus, for a connected graph G, GG(G) is also connected and contains the MST of G. GG(V) can be constructed easily using a localized method with one-hop neighbor information. Gabriel graph was used as a planar subgraph in the face routing protocol [16,24,25] and the GPSR routing protocol [17] that guarantee the delivery of the packet. The same analysis by Bose et al. [11] implied that the length stretch factor of GG(V) is at most $\frac{4\pi\sqrt{2n-4}}{3}$. Recently, Wang et al. [26] showed that the length stretch factor of GG(V) is precisely $\sqrt{n-1}$ actually. Li et al. [3] and Wang [22] then proved that the power stretch factor of any Gabriel graph is one, that is, all power-efficient paths are kept in Gabriel graphs. Therefore, GG(V) is a power spanner but not a length spanner. For unicast routing Gabriel graph is power efficient.

68.2.3 Delaunay Triangulation

While both Gabriel graph and relative neighborhood graph are not length spanners, *Delaunay triangulation* is a well-known length spanner. Assume that there are no four vertices of V that are cocircular. A triangulation of V is a *Delaunay triangulation*, denoted by *Del*(V), if the circumcircle of each of its triangles does not contain any other vertices of V in its interior. A triangle is called the *Delaunay triangle* if its circumcircle is empty of vertices of V. See Figure 68.1(c) for an illustration. Dobkin et al. [27] first proved that the Delaunay Triangulation is a length spanner with length stretch factor bounded by $\frac{1+\sqrt{5}}{2}\pi$. Then Keil and Gutwin [28] improved the constant to be $\frac{4\sqrt{3}}{9}\pi$. Note that the Gabriel graph is a subgraph of the Delaunay triangulation and the power stretch factor of Gabriel graph is one. Thus, the power stretch factor of Delaunay triangulation is also one, due to the monotonic property of power spanner.

Given a set of points V, let *UDel*(V) be the graph of removing all edges of *Del*(V) that are longer than one unit, that is, *UDel*(V) = *Del*(V) ∩ *UDG*(V). Li et al. [29] considered the *unit Delaunay triangulation UDel*(V) for planar spanner of UDG, which is a subset of the Delaunay triangulation. It was proved in Ref. [29] that *UDel*(V) is a $\frac{4\sqrt{3}}{9}\pi$-length-spanner of *UDG*(V).

Though Delaunay triangulation is a well-known planar spanner, it is not appropriate to require the construction of the Delaunay triangulation in the wireless communication environment because of the possible massive communications it requires. Note that the circumcircle of a triangle can be very large (much larger than the transmission range of a wireless node), therefore it may need global information to build the Delaunay triangulation. Recently, several published results [29–31] were proposed to build Delaunay triangulation or its relatives in a localized way. Here we review one of these results.

68.2.4 Local Delaunay Graph

Li et al. [29,30] gave a localized algorithm that constructs a sequence graphs, called *localized Delaunay* $L\,Del^{(k)}(V)$, which are supergraphs of *UDel*(V). Triangle $\triangle uvw$ is called a *k-localized Delaunay triangle* if the interior of the circumcircle of $\triangle uvw$ does not contain any vertex of V that is a k-neighbor of u, v, or w; and all edges of the triangle $\triangle uvw$ have length no more than one unit. The *k-localized Delaunay graph* over a vertex set V, denoted by $LDel^{(k)}(V)$, has exactly all Gabriel edges (edges in GG(V)) and edges of all k-localized Delaunay triangles. The localized algorithm for the construction of $LDel^{(k)}(V)$ goes as follows.

Algorithm 1 CONSTRUCT-$LDel^{(k)}(V)$

```
1: Each node $u$ first gathers the location information of its $k$-hop neighbors $N_k(u)$. It computes the
   Delaunay triangulation $Del(N_k(u))$ of its $k$-neighbors $N_k(u)$, including $u$ itself.
2: For each edge $uv$ of $Del(N_k(u))$, let $\triangle uvw$ and $\triangle uvz$ be two triangles incident on $uv$. Edge $uv$ is a
   Gabriel edge if both angles $\angle uwv$ and $\angle uzv$ are less than $\pi/2$. Node $u$ marks all Gabriel edges $uv$,
   which will never be deleted.
3: Each node $u$ finds all triangles $\triangle uvw$ from $Del(N_k(u))$ such that all three edges of $\triangle uvw$ have length
   at most one unit. If angle $\angle wuv \geq \frac{\pi}{3}$, node $u$ broadcasts a message proposal$(u, v, w)$ to form a
   $k$-localized Delaunay triangle $\triangle uvw$ in $LDel^{(k)}(V)$, and listens to the messages from other nodes.
4: When a node $u$ receives a message proposal$(u, v, w)$, $u$ accepts the proposal of constructing $\triangle uvw$ if
   $\triangle uvw$ belongs to the Delaunay triangulation $Del(N_k(u))$ by broadcasting message accept$(u, v, w)$;
   otherwise, it rejects the proposal by broadcasting message reject$(u, v, w)$.
5: A node $u$ adds the edges $uv$ and $uw$ to its set of incident edges if the triangle $\triangle uvw$ is in the Delaunay
   triangulation $Del(N_k(u))$ and both $v$ and $w$ have sent either accept$(u, v, w)$ or proposal$(u, v, w)$.
```

It was proved that the graph constructed by the above algorithm is $LDel^{(k)}(V)$. Indeed, for each triangle $\triangle uvw$ of $LDel^{(k)}(V)$, one of its interior angles is at least $\pi/3$ and $\triangle uvw$ is in $Del(N_k(u))$, $Del(N_k(v))$ and $Del(N_k(w))$. So one of the nodes among $\{u, v, w\}$ will broadcast the message proposal(u, v, w) to form a k-localized Delaunay triangle $\triangle uvw$. As $Del(N_k(u))$ is a planar graph, and a proposal is made only if $\angle wuv \geq \frac{\pi}{3}$, node u broadcasts at most six proposals. And each proposal is replied by at most two nodes. Therefore, the total communication cost of steps 3–5 is $O(n)$ messages (each message with $\log n$ bits).

As shown in Refs. [29,30], the graph $LDel^{(1)}(V)$ may contain some edges intersecting, while $LDel^{(k)}(V)$ is a planar graph for any $k \geq 2$. Although $LDel^{(1)}(V)$ is not a planar graph, Li et al. [29] proved $LDel^{(1)}(V)$ has thickness 2 which implies it is sparse. Since $UDel(V) \subseteq LDel^{(k)}(V)$ which is proved in Refs. [29,30], $LDel^{(k)}(V)$ is also a length spanner.

For ad hoc networks, we can construct $LDel^{(2)}(V)$, which is guaranteed to be a planar spanner of $UDel(V)$, but it is difficult to collect the 2-hop neighbors for every node in $O(n)$ messages. A total communication cost of a simple broadcast approach to collect 2-hop information is $O(m)$ messages, where m is the number of edges in $UDG(V)$ and could be as large as $O(n^2)$. Recently, Călinescu [32] proposed an approach (using $O(n)$ messages total) that is based on the specific connected dominating set introduced by Alzoubi et al. [33]. Using this approach, Wang and Li [34] proposed an algorithm that can build $LDel^{(2)}$ in $O(n)$ messages, however the constant behind the big-O is still large. To reduce the total communication cost, Li et al. [29,30] do not construct $LDel^{(2)}(V)$, and they extract a planar graph $PLDel(V)$ out of $LDel^{(1)}(V)$ instead. They provided a novel algorithm to make $LDel^{(1)}(V)$ planar using linear communications after building it. The final graph still contains $UDel(V)$ as a subgraph. Thus, it is a spanner of the UDG.

68.3 Bounded-Degree Spanner and Yao's Family

For a topology of ad hoc network, it is also desirable that the node degree in the constructed topology is small and bounded from above by a constant. A small node degree reduces the MAC-level contention and interference, and also may help to mitigate the well-known hidden and exposed terminal problems. In addition, a structure with small degree will improve the overall network throughout [35]. However, all of previous planar topologies are not degree bounded. Thus, in this section, we review several degree bounded spanners.

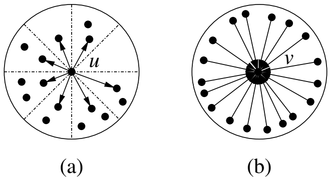


FIGURE 68.2 Illustration of Yao graph where $k = 8$: (a) bound out-degree by Yao structure at node u. (b) unbounded in-degree at node v.

68.3.1 Yao Graph

The Yao graph is proposed by Yao [15] to construct MST of a set of points in high dimensions efficiently. At given node u, any $k \geq 6$ equal-separated rays originated at u define k cones. In each cone, choose the closest node v within the transmission range of u, if there is any, and add a directed link $\overrightarrow{uv}$ (as shown in Figure 68.2[a]). Ties are broken arbitrarily. The remaining edges are deleted from the graph. The resulting directed graph is called the Yao graph, denoted by $\overrightarrow{YG_k}(G)$. If we add the link $\overrightarrow{vu}$ instead of the link $\overrightarrow{uv}$, the graph is denoted by $\overleftarrow{YG_k}(G)$, which is called the *reverse* of the Yao graph. Some researchers used a similar construction named θ-graph [28,36].

The idea of applying Yao structure on $UDG(V)$ to bound node degree is very natural. Hereafter, we use $\overrightarrow{YG_k}(V)$ denote $\overrightarrow{YG_k}(UDG(V))$. It is easy to prove $\overrightarrow{YG_k}(V)$ is connected and sparse. The $\overrightarrow{YG_k}(V)$ has length stretch factor $\frac{1}{1-2\sin\frac{\pi}{k}}$. Thus, its power stretch factor is no more than $(\frac{1}{1-2\sin\frac{\pi}{k}})^{\beta}$. Li et al. [3] proved a stronger result: its power stretch factor is at most $\frac{1}{1-(2\sin\frac{\pi}{k})^{\beta}}$.

Li et al. [2] proposed a structure that is similar to the Yao structure for topology control. Each node u finds a power $p_{u,\alpha}$ such that in every cone of degree α surrounding u, there is some node that u can reach with power $p_{u,\alpha}$. Then the graph G_α contains all edges uv such that u can communicate with v using power $p_{u,\alpha}$. It was proved in Ref. [2] that, if $\alpha \leq \frac{5\pi}{6}$ and the UDG is connected, then graph G_α is a connected graph. On the other hand, if $\alpha > \frac{5\pi}{6}$, they showed that the connectivity of G_α is not guaranteed by giving some counterexamples.

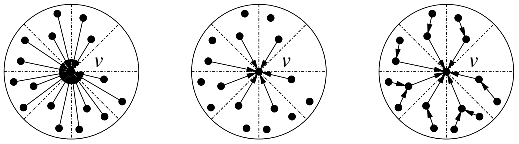
Note that although the directed graph $\overrightarrow{YG_k}(V)$ has a bounded power stretch factor and a bounded out-degree k for each node, some nodes may have very large in-degrees. The nodes configuration given in Figure 68.2(b) will result a very large in-degree for node v. Bounded out-degree gives us advantages when we apply several routing algorithms. However, unbounded in-degree at node v will often cause large overhead at v. Therefore, it is often imperative to construct a sparse network topology such that both the in-degree and the out-degree are bounded by a constant while it is still power efficient.

68.3.2 Symmetric Yao

Symmetric Yao graph, denoted by $YS_k(V)$, was proposed by Wang et al. [7] and Li et al. [37], to bound the node degree at most k. Each node u first applies Yao structure. An edge uv is selected to graph $YS_k(V)$ if and only if both directed edges $\overrightarrow{uv}$ and $\overrightarrow{vu}$ are in the $\overrightarrow{YG_k}(V)$. Then it is obvious that the maximum node degree is k. In Ref. [37], the authors proved that $YS_k(V)$ is strongly connected if $UDG(V)$ is connected and $k \geq 6$. However, it was shown in Ref. [1] recently that $YS_k(V)$ is not a spanner theoretically. They constructed a counterexample to show that $YS_k(V)$ may have large power and length stretch factors.

68.3.3 Sparsified Yao

Another Yao-based algorithm is proposed by Li et al. [4] that constructs a sparser and bounded-degree topology. The basic idea is to apply reverse Yao structure on $\overrightarrow{YG_k}$ to bound the in-degree. Node v chooses a node u from each cone, if there is any, so the directed link $\overrightarrow{uv}$ has the smallest length among all directed links $\overrightarrow{wv}$ in $\overrightarrow{YG_k}$ in that cone (as shown in Figure 68.3[b]). The union of all chosen directed links is the final network topology, denoted by $\overrightarrow{YY_k}(V)$. Notice that in Ref. [1,38], they reinvestigate $\overrightarrow{YY_k}(V)$ structure, and call it *sparsified Yao graph* or *Yao Yao graph.*



(a) (b) (c)

FIGURE 68.3 Illustration of sparsified Yao graph and Yao and sink graph: (a) Large in-degree in YG, star formed by links toward v. (b) YY, bound in-degree by reverse Yao structure. (c) YG*, directed tree $T(v)$ sinked at v.

In Ref. [4], the authors proved $\overrightarrow{YY}_k(V)$ is strongly connected if $UDG(V)$ is connected and $k > 6$. It was proved in Ref. [7] that $\overrightarrow{YY}_k(V)$ is a spanner in civilized UDGs (also called λ-precision UDGs [39]). Here a UDG is a civilized graph if the distance between any two nodes in this graph is larger than a positive constant λ. Li et al. and Wang et al. [4,7] conjectured that $\overrightarrow{YY}_k(V)$ also has constant-bounded length and power stretch factors theoretically in any UDG. Recently, Jia et al. [40] and Schindelhauer et al. [41] proved that $\overrightarrow{YY}_k(V)$ has a constant-bounded power stretch factor theoretically. However, it is still an open problem whether it is a length spanner.

68.3.4 Yao and Sink Structure

Arya et al. [9] gave an ingenious technique to generate a bounded-degree graph with constant length stretch factor. Li et al. [3]; applied the same technique to construct a sparse network topology with a bounded degree and a bounded power stretch factor from $\overrightarrow{YG}_k(V)$. The technique is to replace the directed star consisting of all links toward a node v in $\overrightarrow{YG}_k(V)$ (as shown in Figure 68.3[a]) by a directed tree $T(v)$ of a bounded degree with v as the sink (as shown in Figure 68.3[c]). Tree $T(v)$ is constructed recursively. The algorithm is as follows. First, construct the graph $\overrightarrow{YG}_k(V)$. Each node v will have a set of incoming nodes $I(v) = \{u \mid \overrightarrow{uv} \in \overrightarrow{YG}_k(V)\}$. For each node v, use the following algorithm Tree$(v, I(v))$ to build tree $T(v)$.

Algorithm 2 CONSTRUCT-$T(v)$ TREE$(v,I(v))$

```
To partition the unit disk centered at $v$, choose $k$ equal-sized cones centered at $v$: $C_1(v)$, $C_2(v)$, ..., $C_k(v)$.
Node $v$ finds the nearest node $y_i \in I(v)$ in $C_i(v)$, for $1 \leq i \leq k$, if there is any. Link $\overrightarrow{y_i v}$ is added to $T(v)$ and $y_i$ is removed from $I(v)$. For each cone $C_i(v)$, if $I(v) \cap C_i(v)$ is not empty, call Tree$(y_i, I(v) \cap C_i(v))$ and add the created edges to $T(v)$.
```

The union of all trees $T(v)$ is called the *sink structure* $\overrightarrow{YG}^*_k(V)$. Notice that, node v constructs the tree $T(v)$ and then broadcasts the structure of $T(v)$ to all nodes in $T(v)$. Since the total number of edges in the Yao structure is at most $k \cdot n$, the total number of edges of $T(v)$ of all nodes v is also at most $k \cdot n$. Thus, the total communication cost of broadcasting the $T(v)$ to all its neighbors is still at most $k \cdot n$. The algorithm uses a directed tree $T(v)$ to replace the directed star for each node v. Therefore, if nodes u and v are connected by a path in $\overrightarrow{YG}_k$, they are also connected by a path in $\overrightarrow{YG}^*_k$. It is already known that $\overrightarrow{YG}_k$ is strongly connected if $UDG(V)$ is connected, so does $\overrightarrow{YG}^*_k$. Li et al. [3] also proved that the power stretch factor of $\overrightarrow{YG}^*_k(V)$ is at most $(\frac{1}{1-(2\sin\frac{\pi}{k})^\beta})^2$ and the maximum degree of $\overrightarrow{YG}^*_k(V)$ is at most $(k+1)^2 - 1$.

68.3.5 Ordered Yao

Bose et al. [42]; study a variant of θ-graphs called *ordered θ-graphs*. An ordered θ-graph of V is obtained by inserting the points of V in some order. When a point p is inserted, we draw the same cones around p and connect p to its closest previously inserted neighbor in each cone. An ordered θ-graph of V is dependent on the order imposed on V; different orderings of V can produce different graphs. Nevertheless, in Ref. [42], they show that ordered θ-graphs are also spanners, regardless of the ordering used.

In the same way, we can generally define an *ordered Yao graph* $YO_k(V)$ by some order π imposed on V. Using same arguments in Ref. [42], we can prove that the power stretch factor of the π-ordered Yao graph

$YO_k(V)$ is at most $\frac{1}{1-(2\sin\frac{\pi}{k})^{\beta}}$, for any ordering π. However, the node degree of $YO_k(V)$ is *not* bounded by k, since for node u after applying Yao structure, other node can still add more edges to node u. The communications cost of this algorithm is $O(n)$, if an ordering is given. Note that we can use a local order π' instead of the global order π. Since we can use $O(n)$ message to build a local order, like we do in the localized algorithm in next section, $YO_k(V)$ can be built using $O(n)$ messages.

Note that in ordered Yao graph, when you process a node, it only considers all previous processed nodes. If we change it to only consider all unprocessed nodes, the spanner proof still holds.

68.4 Bounded-Degree Planar Spanner

The structures discussed so far either have bounded degree or are planar or spanners, but none of the structures have all these three properties together. We then review some recent results that can locally construct a bounded-degree planar spanner for ad hoc networks.

68.4.1 Delaunay Triangulation Plus Yao Structure

Bose et al. [43] proposed a centralized $O(n \log n)$-time algorithm that constructs a plan t-spanner for a given nodes set V, for $t \simeq 10.02$, such that the node degree is bounded from above by 27. As we knew, this algorithm is the first method to compute a plane spanner of bounded degree. However, their method is impossible to have a localized even distributed version, since they use breadth-first-search (BFS) and many operations on polygons (such as degree-3 partitions). Note that may take $O(n^2)$ communications.

Inspired by Bose et al.'s method, Li and Wang [44] recently proposed a centralized algorithm for building a planar spanner with bounded node degree for $UDG(V)$. The basic idea of their method is to combine Delaunay triangulation and the ordered Yao structure [42]. The algorithm is as follows. Here, we assume each node u has a unique ID denoted by $ID(u)$.

Algorithm 3 CENTRALIZED ALGORITHM: CONSTRUCT BOUNDED-DEGREE PLANAR LENGTH SPANNER

```
1: Compute Delaunay triangulation $Del(V)$ and remove the edges whose length is longer than 1 in
   $Del(V)$. Call the remaining graph unit Delaunay triangulation $UDel(V)$. For every node $u$, we know
   its unit Delaunay neighbors $N_{UDel}(u)$ and its node degree $d_u$ in $UDel(V)$.
2: Find an order $\pi$ of $V$ as follows: Let $G_1 = UDel(V)$ and $d_{G,u}$ is the node degree of $u$ in graph $G$.
   Remove the node $u$ with the smallest value of $(d_{G_i,u}, ID(u))$ from $G_i$, let $\pi_u = n - i + 1$, and call
   the remaining graph $G_{i+1}$. Repeat this procedure for $1 \leq i \leq n$. Obviously, in ordering $\pi$, node $u$ at
   most has five edges to its predecessors $P_u$ in $UDel(V)$. Here, $x$ is a predecessor of $y$ if $\pi_x < \pi_y$.
3: Let $E$ and $E'$ be the edge sets of $UDel(V)$ and the desired spanner. Initialize $E' = \emptyset$ and all nodes in
   $V$ are unprocessed. Then, same with the algorithm for point set, for each node $u$ in $V$, following the
   increasing order $\pi$, run the following steps to add some edges to $E'$:
    (a) Node $u$ uses its predecessors (processed Unit Delaunay neighbors) in $E$ to define at most five
        open sectors at node $u$ (assume $v_1, \ldots, v_5$ are the processed neighbors of node $u$ in $UDel(V)$).
        For each sector, we divide it into a minimum number of open cones of degree $\alpha$, where $\alpha \leq \pi/3$.
    (b) For each cone, let $s_1, s_2, \ldots, s_m$ be the ordered neighbors $N_{UDel}(u)$ of $u$ in this cone. That is,
        $s_1, s_2, \ldots, s_m$ are all unprocessed nodes that are connected by an edge of the unit Delaunay
        triangulation to $u$. For each cone, first add the shortest edge in $E$ that is adjacent to $u$ to the
        edge set $E'$, then add to $E'$ all the edges $s_j s_{j+1}$ between its geometrically ordered unprocessed
        neighbors in this cone, $1 \leq j < m$. Note that, here such edges $s_j s_{j+1}$ are not necessarily
        in $UDel(V)$. For example, when node $u$ has a Delaunay neighbor $x$ such that $ux$ intersects
        edge $s_i s_{i+1}$ and $\|ux\| > 1$.
    (c) Mark node $u$ as processed.
   Repeat this procedure in order of $\pi$, until all nodes are processed. Let $BPS_c(V)$ denote the final graph
   formed by edge set $E'$.
```

Note that the algorithm uses *open* sectors, which means that in the algorithm we do not consider adding the edges on the boundaries (any edge involved previously processed neighbors). In other words, the open cones do not include any edges uv_i. This guarantees the algorithm does not add any edges to node v_i after v_i has been processed. This approach bounds the node degree. In Ref. [44], the authors proved that the maximum node degree of the graph $BPS_c(V)$ is at most $19 + \lceil \frac{2\pi}{\alpha} \rceil$. For example, when $\alpha = \pi/3$, the maximum node degree is at most 25. In Ref. [44], the authors also proved that graph $BPS_c(V)$ is a planar t-length-spanner, where $t = \max\{\frac{\pi}{2}, \pi \sin \frac{\alpha}{2} + 1\} \cdot C_{del}$. Hereafter, we use C_{del} to denote the length stretch factor of the Delaunay triangulation.

68.4.2 Local Delaunay Graph Plus Yao Structure

By using local Delaunay graph and local ordering, Wang and Li [34] converted their centralized method to an efficient localized algorithm for building bounded degree planar spanner.

Algorithm 4 LOCALIZED ALGORITHM: CONSTRUCT BOUNDED-DEGREE PLANAR LENGTH SPANNER

```
1: First, compute the planar localized Delaunay triangulation $L\,Del^{(2)}(V)$ (using the method in Ref. [32]
   to collect the location information of $N_2(u)$), so that every node $u$ knows all its neighbors $N_{L\,Del^{(2)}}(u)$
   and its node degree $d(u)$ in $L\,Del^{(2)}(V)$. Assume a synchronized method is used to collect $N_{L\,Del^{(2)}}(u)$
   for every node $u$.
2: Build a local order $\pi$ of $V$ as follows: (Every node $u$ initializes $\pi_u = 0$, i.e., unordered.)
   (a) If node $u$ has $\pi_u = 0$ and $d(u) \leq 5$, then $u$ queries[1] each node $v$, from its unordered neighbors,
       the current degree $d(v)$. If node $u$ has the smallest ID among all unordered neighbors $v$ with
       $d(v) \leq 5$, node $u$ sets
           $$\pi_u = \max\{\pi_v \mid v \in N_{L\,Del^{(2)}}(u)\} + 1$$
       and broadcasts $\pi_u$ to its neighbors $N_{L\,Del^{(2)}}(u)$.
   (b) If node $u$ receives a message from its neighbor $v$ saying that $\pi_v = k$, it updates its $d(u) = d(u) - 1$
       and also updates the order $\pi_v$ stored locally. So $d(u)$ represents how many neighbors are not
       ordered so far.
       If node $u$ finds that $d(u) \leq 5$ and $\pi_u = 0$, it goes to step 2 (a).
       When node $u$ finds that $d(u) = 0$ and $\pi_u > 0$, it can go to step 3.
3: Build structures based on local order $\pi$ as follows: (Initialize all nodes unprocessed)
   (a) If an unprocessed node $u$ has the highest local order in its unprocessed neighbors $N_u$ in
       $L\,Del^{(2)}(V)$, let $k$ be the number of processed neighbors[2] of $u$ in $L\,Del^{(2)}(V)$. Assume that
       $v_1, v_2, \ldots, v_k$ be the processed neighbors of $u$ in $L\,Del^{(2)}(V)$. Node $u$ divides its transmission
       range into $k$ open sectors cut by the rays from $u$ to these processed neighbors. Then divide each
       sector into a minimum number of open cones of degree at most $\alpha$ with $\alpha \leq \pi/3$. For each cone,
       let $s_1, s_2, \ldots, s_m$ be the ordered unprocessed neighbors of $u$ in $N_{L\,Del^{(2)}}(u)$. For this cone, node
       $u$ first adds an edge $us_i$, where $s_i$ is the nearest neighbor among $s_1, s_2, \ldots, s_m$. Node $u$ then
       tells $s_1, s_2, \ldots, s_m$ to add all the edges $s_j s_{j+1}$, $1 \leq j < m$. Node $u$ marks itself processed, and
       tells all nodes in $N_{L\,Del^{(2)}}(u)$ that it is processed.
   (b) If an unprocessed node $v$ receives a message for adding edge $vv'$ from its neighbor $u$, it adds
       edge $vv'$.
4: When all nodes are processed, the final network topology is denoted by $BPS(V)$.
```

[1] If some unordered neighbor with $d(v) \leq 5$ has smaller ID, we call such query round a *failed round.* Node u performs a new round of queries only if it finds that the number of its unordered neighbors has been reduced ($d(u)$ has reduced in step 2[b]). So there are at most five rounds of queries.

[2] There are at most five processed neighbors since graph $L\,Del^{(2)}(V)$ is planar.

Note that the ordering computed by this method is not a global ordering. Some nodes may have the same order. However, no two neighboring nodes in $LDel^{(2)}(V)$ receive the same order. Thus, after all nodes are ordered, the algorithm will process all nodes. Observe that the algorithm does not process two neighboring nodes at the same time. Assume that there are two nodes, say u and v are processed at the same time. Remember that the algorithm processes a node only if it has the highest ordering among its unprocessed neighbors. Thus, nodes u and v must receive the same order, that is, $\pi_u = \pi_v$, which is impossible in the ordering method. In Ref. [34], the authors also proved that graph $BPS(V)$ is a planar t-spanner, where $t = \max\{\frac{\pi}{2}, \pi \sin\frac{\alpha}{2} + 1\} \cdot C_{del}$. The proofs of the planar and spanner properties are much complex than the centralized ones, refer to Ref. [34] for details. In addition, Algorithm 4 uses at most $O(n)$ messages, where each message has $O(\log n)$ bits. However, the hidden constant could be as high as several hundreds since the method needs to collect the 2-hop information for every node.

68.4.3 Gabriel Graph Plus Yao Structure

Remember that Gabriel graph is a planar power spanner. To reduce the total communication cost, Song et al. [45] proposed new methods by applying the ordered Yao structures on Gabriel graph to bound node degree. Note that the Gabriel graph is much simpler and easier to build than localized Delaunay graph. The algorithm is as follows.

Algorithm 5 Localized Algorithm 1: Construct Bounded-Degree Planar Power Spanner

```
1: Each node self-constructs the Gabriel graph $GG$ locally. Let $N_{GG}(u)$ be the neighbors set of node $u$ in $GG$.
2: Each node $u$ decides its order $\pi$ locally using the same method in Algorithm 4 from Ref. [34].
3: All nodes self-form the final topology based on local order $\pi$ as follows. Initially, all nodes are marked with White color, that is, unprocessed. Let $N_{OYGG}(u)$ be the set of neighbors of $u$ in the final topology, which is initialized as $N_{GG}(u)$.
     (a) If node $u$ is unprocessed (marked White), and it has the largest order $\pi[u]$ among all its White neighbors in $N_{GG}(u)$, it divides its transmission range (which is a unit disk centered at the node $u$) into $k$ equal-sized cones, keeps one nearest White neighbor $v \in N_{OYGG}(u)$ (if available) in each cone and deletes others. Node $u$ marks itself Black, that is, processed, and notifies all nodes in $N_{GG}(u)$ of the deleted edges through a broadcasting message UpdateN. The message UpdateN includes all unselected neighbors.
     (b) Once the node $u$ receives the message UpdateN for deleting edge $vu$ from its neighbor $v$, it deletes the node $v$ from its local list $N_{OYGG}(u)$.
   When all nodes are processed, all the remaining edges $\{uv | v \in N_{OYGG}(u)\}$ form the final network topology OrdYaoGG.
```

It is proved in Ref. [45] that *OrdYaoGG* is a bounded-degree planar power spanner. The power stretch factor is at most $\frac{1}{1-(2\sin\frac{\pi}{k})^{\beta}}$ while the node degree is bounded from above by a positive constant $k + 5$ where $k > 6$ is an adjustable parameter. Moreover, they showed that the structure can be constructed using at most $24n$ messages, where each message is $O(\log n)$ bits.

Furthermore, in the same paper, the authors proposed another method to build a degree-bounded planar power spanner, which can be constructed easier and demands less communication cost during construction.

Algorithm 6 LOCALIZED ALGORITHM 2: CONSTRUCT BOUNDED-DEGREE PLANAR POWER SPSNNER

```
First, each node self-constructs the Gabriel graph $GG$ locally.
All nodes together self-form the final topology as follows. Initially, each node $u$ is marked with WHITE color, that is, unprocessed, and initializes $N_{SYGG}(u)$ as the set of all the neighbor nodes in $GG$.
    (a) If a WHITE node $u$ has the smallest ID among its WHITE neighbors in $GG$, it divides its transmission range into $k$ equal-sized cones where $k > 8$ is an adjustable parameter. In each cone, node $u$ checks whether there are some BLACK nodes in $N_{SYGG}(u)$ within same cone:
        i. Yes. Node $u$ keeps the closest BLACK neighbor $v \in N_{SYGG}(u)$ among them and deletes all the other links in the cone.
        ii. No. Node $u$ keeps a closest WHITE neighbor $v \in N_{SYGG}(u)$ (if available) among them and deletes all the other links in the cone.
        After processing all $k$ cones, node $u$ marks itself BLACK, that is, processed, then notifies each deleted neighboring node $v$ in $GG$ by a broadcasting message UPDATEN.
    (b) Once a WHITE node $v$ receives the message UPDATEN from a neighbor $u$ in $GG$, it checks whether it is itself in the nodes set for deleting: if so, it deletes the sending node $u$ from $N_{SYGG}(v)$, otherwise, marks $u$ as BLACK in its local list $N_{SYGG}(v)$.
    (c) Once a BLACK node $v$ receives the message UPDATEN from a neighbor belonging to $N_{SYGG}(v)$, it checks whether it is itself in the nodes set for deleting: if so, it deletes the sending node $u$ from $N_{SYGG}(v)$, otherwise, marks $u$ as BLACK in its local list $N_{SYGG}(v)$.
When all nodes are processed, all selected edges $\{uv | v \in N_{SYGG}(u)\}$ form the final network topology, denoted by SYaoGG.
```

Algorithm 6 further reduces the communication cost during constructing a degree-bounded planar power spanner to $3n$ messages, because it does not demand the local ordering before construction. It also reduces the degree bound to k, and keeps all other nice properties, except that the theoretical power stretch factor is relaxed to $\frac{\sqrt{2}^{\beta}}{1-(2\sqrt{2}\sin\frac{\pi}{k})^{\beta}}$, where $k > 8$ is an adjustable parameter.

Note that both *OrdYaoGG* and *SYaoGG* are degree-bounded planar power spanners, but they are not length spanners. However, *BPS*(V) is a degree-bounded planar length spanner.

68.5 Other Spanners

Beside the spanners (planar spanner, bounded-degree spanner, and bounded-degree planar spanner) we reviewed above, there are many other spanners that have been studied for ad hoc network applications.

Fault-tolerant geometric spanners have been studied heavily [14,46–48], but most of the solutions are centralized methods which are too complex to have localized versions. Lukovszki et al. [49] gave a method based on θ graph to construct a spanner that can sustain k-nodes or links failures for complete graph. Similarly, Li et al. [50] also proposed a method based on Yao structure to build a fault-tolerant spanner. Bahramgiri et al. [51] generalized the cone-based local heuristic of Wattenhofer et al. [2,8] to achieve the fault tolerance. We can prove that their resulted graph is also a length spanner (the proof is similar to Ref. [50]).

Li et al. [52] proposed a new bounded-degree planar spanner which is also low weighted. Here a topology is low weighted if the total link length of the topology is within a constant factor of that of MST. For ad hoc networks, low-weighted structures enable energy-efficient broadcasting.

Burkhart et al. [53] studied the spanners which can effectively constrain interference. Following this direction, Li et al. [54] and Moscibroda and Wattenhofer [55] also studied more general low interference topologies for ad hoc networks.

TABLE 68.1 Summary of the Spanners

	Power Spanner	Length Spanner	Bounded Degree	Planar	Local Construction
RNG	No	No	No	Yes	Yes
GG	Yes	No	No	Yes	Yes
Del	Yes	Yes	No	Yes	No
LDel	Yes	Yes	No	Yes	Yes
YG	Yes	Yes	No	No	Yes
*YG**	Yes	Yes	Yes	No	Yes
YY	Yes	Open	Yes	No	Yes
YS	No	No	Yes	No	Yes
YO	Yes	Yes	No	No	Yes
BPS	Yes	Yes	Yes	Yes	Yes
OYAOGG	Yes	No	Yes	Yes	Yes
SYAOGG	Yes	No	Yes	Yes	Yes

Schindelhauer et al. [56,41] defined a new concept *weak spanner* in which the path can be arbitrarily longer than the original one but must remain within a disk or sphere of radius constant times the Euclidean distance between two vertices. They studied the relationship among length spanner, weak spanner, and power spanner. In addition, they proved that sparsified Yao graph is weak spanner and power spanner.

The spanners reviewed in this chapter are spanners for UDG where each node has same transmission range. However, practically, the networks are never so perfect as UDGs. Kuhn et al. [57] modeled ad hoc networks as *quasi UDG* and gave a spanner for quasi UDG. Li et al. [58] also studied spanners for networks with nonuniform transmission ranges where different node could have different transmission ranges. They extended the Yao graph, sparsified Yao graph, and Yao and sink graphs to the nonuniform case.

68.6 Conclusion

In this chapter, we reviewed several methods to construct geometric spanners as network topologies for ad hoc networks which can approximate the underlying communication graph well. The summary of properties of these spanners is given in Table 68.1. In this chapter, we did not give the detailed algorithms, proofs, and simulation results, please refer to the reference for more detail.

References

[1] Grünewald, M., Lukovszki, T., Schindelhauer, C., and Volbert, K., Distributed maintenance of resource efficient wireless network topologies, *Proc. 8th European Conf. on Parallel Comput.*, Paderborn, Germany, 2002.

[2] Li, L., Halpern, J. Y., Bahl, P., Wang, Y.-M., and Wattenhofer, R., Analysis of a cone-based distributed topology control algorithms for wireless multi-hop networks, *Proc. ACM Symp. on Principle of Dist. Comput.*, Newport, Rhode Island, USA, 2001.

[3] Li, X.-Y., Wan, P.-J., and Wang, Y., Power efficient and sparse spanner for wireless ad hoc networks, *Proc. IEEE Int. Conf. on Computer Comm. and Networks*, Scottsdale, AZ, USA, 2001, p. 564.

[4] Li, X.-Y., Wan, P.-J., Wang, Y., and Frieder, O., Sparse power efficient topology for wireless networks, *Proc. IEEE Hawaii Int. Conf. on System Sciences*, Big Island, HI, USA, 2002.

[5] Rajaraman, R., Topology control and routing in ad hoc networks: a survey, *SIGACT News*, 33, 60, 2002.

[6] Ramanathan, R. and Hain, R., Topology control of multihop wireless networks using transmit power adjustment, *Proc. IEEE INFOCOM*, Tel Aviv, Israel, Vol. 2, 2000, p. 404.

[7] Wang, Y., Li, X.-Y., and Frieder, O., Distributed spanner with bounded degree for wireless networks, *Int. J. of Foundations of Computer Science*, 14(2), 183, 2003.

[8] Wattenhofer, R., Li, L., Bahl, P., and Wang, Y.-M., Distributed topology control for wireless multihop ad-hoc networks, *Proc. IEEE INFOCOM*, Anchorage, Alaska, USA, 2001.

[9] Arya, S., Das, G., Mount, D., Salowe, J., and Smid, M., Euclidean spanners: short, thin, and lanky, *Proc. ACM STOC*, Las Vegas, Nevada, USA, 1995, p. 489.

[10] Arya, S. and Smid, M., Efficient construction of a bounded degree spanner with low weight, *Proc. European Symp. Algorithms*, Lecture Notes in Computer Science, Utrecht, The Netherlands, Vol. 855, 1994, p. 48.

[11] Bose, P., Devroye, L., Evans, W., and Kirkpatrick, D., On the spanning ratio of Gabriel graphs and beta-skeletons, *Proc. Latin American Theor. Informatics,* Cancun, Mexico, 2002.

[12] Chandra, B., Das, G., Narasimhan, G., and Soares, J., New sparseness results on graph spanners, *Proc. ACM Symp. on Computational Geom.*, Berlin, Germany, 1992, p. 192.

[13] Karavelas, M. I. and Guibas, L. J., Static and kinetic geometric spanners with applications, *Proc. ACM/SIAM SODA*, Washington, DC, USA, 2001, p. 168.

[14] Levcopoulos, C., Narasimhan, G., and Smid, M., Efficient algorithms for constructing fault-tolerant geometric spanners, *Proc. ACM STOC*, Dallas, Texas, USA, 1998.

[15] Yao, A. C.-C., On constructing minimum spanning trees in k-dimensional spaces and related problems, *SIAM J. Comput.*, 11, 721, 1982.

[16] Bose, P., Morin, P., Stojmenovic, I., and Urrutia, J., Routing with guaranteed delivery in ad hoc wireless networks, *ACM/Kluwer Wireless Networks*, 7(6), 609–616, 2001.

[17] Karp, B. and Kung, H., GPSR: greedy perimeter stateless routing for wireless networks, *Proc. ACM Int. Conf. on Mobile Computing and Networking*, Boston, Massachusetts, United States, 2000.

[18] Kuhn, F., Wattenhofer, R., and Zollinger, A., Asymptotically optimal geometric mobile ad-hoc routing, *Proc. 6th ACM Int. Workshop on Disc. Algorithms and Methods for Mobile Comput. and Comm.*, Atlanta, Georgia, USA, 2002, p. 24.

[19] Jaromczyk, J. and Toussaint, G., Relative neighborhood graphs and their relatives, *Proc. IEEE*, 80(9), 1502, 1992.

[20] Toussaint, G. T., The relative neighborhood graph of a finite planar set, *Pattern Recognition*, 12(4), 261, 1980.

[21] Seddigh, M., Gonzalez, J. S., and Stojmenovic, I., RNG and internal node based broadcasting algorithms for wireless one-to-one networks, *ACM Mobile Comput. and Comm. Rev.*, 5(2), 37, 2002.

[22] Wang, Y., Efficient Localized Topology Control for Wireless Ad Hoc Networks, Ph.D. thesis, Department of CS, Illinois Institute of Technology, 2004.

[23] Gabriel, K. and Sokal, R., A new statistical approach to geographic variation analysis, *Syst. Zoolo.*, 18, 259, 1969.

[24] Datta, S., Stojmenovic, I., and Wu, J., Internal node and shortcut based routing with guaranteed delivery in wireless networks, *Cluster Comput.*, 5(2), 169, 2002.

[25] Stojmenovic, I. and Datta, S., Power and cost aware localized routing with guaranteed delivery in wireless networks, *Proc. IEEE Symp. on Computers and Comm.*, Taormina, Italy, 2002.

[26] Wang, W., Li, X.-Y., Moaveni-Nejad, K., Wang, Y., and Song, W.-Z., The spanning ratios of beta-skeletons, *Proc. Canadian Conf. on Computational Geom.*, Halifax, Canada, 2003.

[27] Dobkin, D., Friedman, S., and Supowit, K., Delaunay graphs are almost as good as complete graphs, *Disc. Comput. Geom.*, 5(4), 399–407, 1990.

[28] Keil, J. M. and Gutwin, C. A., Classes of graphs which approximate the complete euclidean graph, *Disc. Comput. Geom.*, 7(1), 13–28, 1992.

[29] Li, X.-Y., Calinescu, G., and Wan, P.-J., Distributed construction of planar spanner and routing for ad hoc wireless networks, *Proc. IEEE INFOCOM*, New York, USA, 2002.

[30] Li, X.-Y., Calinescu, G., Wan, P.-J., and Wang, Y., Localized Delaunay triangulation with application in wireless ad hoc networks, *IEEE Trans. Parallel Dist. Processing*, 14(10), 1035, 2003.

[31] Gao, J., Guibas, L. J., Hershberger, J., Zhang, L., and Zhu, A., Geometric spanner for routing in mobile networks, *Proc. ACM Symp. on Mobile Ad Hoc Networking and Comput.*, Long Beach, California, USA, 2001.

[32] Călinescu, G., Computing 2-hop neighborhoods in ad hoc wireless networks, *Proc. Int. Conf. on Ad-Hoc, Mobile, and Wireless Networks, AdHoc-Now*, Montreal, Canada, 2003.

[33] Alzoubi, K. M., Wan, P.-J., and Frieder, O., Message-optimal connected dominating sets in mobile ad hoc networks, *Proc. ACM Int. Symp. on Mobile Ad Hoc Networking & Computing*, Lausanne, Switzerland, 2002.

[34] Wang, Y., and Li, X.-Y., Localized construction of bounded degree and planar spanner for wireless ad hoc networks, *Mobile Networks and Applications*, 11(2), 161–175, 2006.

[35] Kleinrock, L. and Silvester, J., Optimum transmission radii for packet radio networks or why six is a magic number, *Proc. IEEE National Telecommunications Conf.*, Birmingham, AL, USA, 1978, p. 431.

[36] Lukovszki, T., New Results on Geometric Spanners and Their Applications, Ph.D. thesis, University of Paderborn, 1999.

[37] Li, X.-Y., Stojmenovic, I., and Wang, Y., Partial Delaunay triangulation and degree limited localized Bluetooth multihop scatternet formation, *IEEE Trans. Parallel Dist. Syst.*, 15(4), 350, 2004.

[38] Rührup, S., Schindelhauer, C., Volbert, K., and Grünewald, M., Performance of distributed algorithms for topology control in wireless networks, *Proc. Int. Parallel and Dist. Processing Symp.*, Nice, France, 2003.

[39] Hunt, H. B., III, Marathe, M. V., Radhakrishnan, V., Ravi, S. S., Rosenkrantz, D. J., and Stearns, R., NC-approximation schemes for NP- and PSPACE -hard problems for geometric graphs, *J. Algorithms*, 26(2), 238, 1998.

[40] Jia, L., Rajaraman, R., and Scheideler, C., On local algorithms for topology control and routing in ad hoc networks, *Proc. ACM Symp. on Parallel Algorithms and Architectures*, San Diego, California, USA, 2003.

[41] Schindelhauer, C., Volbert, K., and Ziegler, M., Geometric spanners with applications in wireless networks, *Comput. Geometry: Theor. Appl.*, 36(3), 197–214, 2007.

[42] Bose, P., Gudmundsson, J., and Morin, P., Ordered theta graphs, *Proc. Canadian Conf. on Computational Geom.*, Lethbridge, Alberta, Canada, 2002.

[43] Bose, P., Gudmundsson, J., and Smid, M., Constructing plane spanners of bounded degree and low weight, *Proc. European Symp. on Algorithms*, Rome, Italy, 2002.

[44] Li, X.-Y. and Wang, Y., Efficient construction of low weight bounded degree planar spanner, *Int. J. of Comput. Geom. and Appl.*, 14(1–2), 69, 2004.

[45] Song, W.-Z., Wang, Y., Li, X.-Y., and Frieder, O., Localized algorithms for energy efficient topology in wireless ad hoc networks, *Proc. ACM Int. Symp. on Mobile Ad Hoc Networking and Computing*, Tokyo, Japan, 2004.

[46] Levcopoulos, C., Narasimhan, G., and Smid, M., Improved algorithms for constructing fault tolerant geometric spanners, *Algorithmica*, 32(1), 144–156, 2002.

[47] Levcopoulos, C., Narasimhan, G., and Smid, M., Efficient algorithms for constructing fault-tolerant geometric spanners, *Proc. ACM STOC*, Dallas, Texas, USA, 1998, p. 186.

[48] Czumaj, A. and Zhao, H., Fault-tolerant geometric spanners, *Proc. ACM Conf. on Computational Geom.*, San Diego, California, USA, 2003, p. 1.

[49] Lukovszki, T., New results on fault tolerant geometric spanners, *Workshop on Algorithms and Data Structures*, Vancouver, Canada, 1999, p. 193.

[50] Li, X.-Y., Wan, P.-J., Wang, Y., Yi, C.-W., and Frieder, O., Robust deployment and fault tolerant topology control for wireless ad hoc networks, *J. Wireless Comm. Mobile Comput.*, 4(1), 109, 2004.

[51] Bahramgiri, M., Hajiaghayi, M. T., and Mirrokni, V. S., Fault-tolerant and 3-dimensional distributed topology control algorithms in wireless multi-hop networks, *Proc. IEEE Int. Conf. on Computer Comm. and Networks*, Miami, Florida, USA, 2002, p. 392.

[52] Li, X.-Y., Song, W.-Z., and Wang, W., A unified energyefficient topology for unicast and broadcast, *Proc. ACM Int. Conf. on Mobile Computing and Networking*, Cologne, Germany, 2005.

[53] Burkhart, M., von Rickenbach, P., Wattenhofer, R., and Zollinger, A., Does topology control reduce interference? *Proc. 5th ACM Int. Symp. on Mobile Ad Hoc Networking and Comput.*, Tokyo, Japan, 2004.

[54] Li, X.-Y., Moaveni-Nejad, K., Song, W.-Z., and Wang, W., Interference-aware topology control for wireless sensor networks, *Proc. 2nd IEEE Conf. on Sensor and Ad Hoc Comm. and Networks*, Santa Clara, California, USA, 2005.

[55] Moscibroda, T. and Wattenhofer, R., Minimizing interference in ad hoc and sensor networks, *Proc. ACM Joint Workshop on Foundations of Mobile Computing*, Cologne, Germany, 2005.

[56] Schindelhauer, C., Volbert, K., and Ziegler, M., Spanners, weak spanners, and power spanners for wireless networks, *Proc. Int. Symp. on Algorithms and Computation*, Hong Kong, China, 2004.

[57] Kuhn, F., Wattenhofer, R., and Zollinger, A., Ad-hoc networks beyond unit disk graphs, *Proc. ACM Joint Workshop on Foundations of Mobile Computing*, San Diego, CA, USA, 2003.

[58] Li, X.-Y., Song, W.-Z., and Wang, Y., Efficient topology control for wireless ad hoc networks with non-uniform transmission ranges, *Wireless Networks*, 11(3), 255, 2005.

69

Multicast Topology Inference and Its Applications

Hui Tian
University of Science and Technology of China

Hong Shen
University of Adelaide

69.1 Introduction

As networks continue to grow explosively both in size and internal complexity, the ever increasing tremendous traffic load and applications drive researchers to develop techniques for analyzing network performance and managing network resources for optimal usage. All these tasks require that end-systems know the internal network characteristics. Discovery of internal information such as topology and localized lossy links plays an important role in resource management, loss recovery, and congestion control [1–3]. As a result of this different techniques have been developed to identify the internal characteristics of networks.

Existing approaches to discover the internal characteristics of networks are classified into three types: (i) setting probes in the network, collecting statistics at internal nodes periodically, and generating topology reports or link-level performance; (ii) characterizing the network based on end-to-end behavior of point-to-point traffic such as that generated by TCP or UDP; and (iii) inferring link-level loss behavior and topology from end-to-end measurements.

The first approach requires support from internal nodes. It takes time and extra equipment or software to collect and analyze data. Usually only authorized network administrator and developer can perform these tasks. This approach can neither be made to update the network internal information in time nor is it scalable. The second approach has the same problem as the first one. The only difference is that it is performed on the links of the network. Compared with the former two approaches, the third one is the most intelligent approach. It infers network internal characteristics by using multicast traffic from end-to-end measurements, without needing the cooperation of internal nodes. This is a branch of network tomography, which has attracted considerable attention in recent years because the end-based inference approach is much more practical and scalable than previous approaches. Multicast probe makes the approach easier and more efficient.

A crucial step in the identification of the network internal characteristics for the end-based inference approach is the identification of the network topology. Topology inference research from end-to-end measurements can be found in Refs. [3–8]. The key idea underlying the approach on multicast measurements is that receivers sharing common paths on the multicast tree associated with a given source will see a correlation in their packet losses or delays. The multicast topology and internal loss and delay performance can thus be inferred based on the shared loss or delay statistics of the probe packets transmitted. However, the prevalent methods to estimate correlation used in Refs. [4–7] for siblings identification may produce fault results as pointed out in Ref. [9]. To overcome this difficulty, we present our Hamming distance-based classification approach [10], which is based on the Hamming distance of sequences on receipt/loss of probe packets maintained at each pair of sibling nodes. In comparison with other approaches, our approach is simpler and more effective. On the basis of the topology identified, we also discuss the applications of the Hamming distance approach for the inference of other internal characteristics such as internal delay and link loss performance [9,11]. The efficiency and effectiveness of our inferences using the Hamming distance approach are validated by the simulation results discussed in this chapter.

The chapter is organized as follows. In Section 69.2 two mathematical models to characterize multicast networks are introduced. Section 69.3 describes the Hamming distance classification approach and presents our Binary Hamming distance Classification-based algorithm (BHC) for multicast topology inference. Section 69.4 presents the Hamming distance matrix approach for internal delay and link loss performance analysis. Section 69.5 shows the application of Hamming distance for general tree topology inference. Section 69.6 concludes the chapter. The contents of this chapter are based on our previous work reported in Refs. [9–11].

69.2 Mathematical Models of Multicast Network

We use the multicast tree model and the loss/delay model which have been widely used [5–7,12,13] to study the internal characteristics of networks. We have made a number of assumptions. Routing and network topology are assumed to be static during the measurement period, as dynamic routing and topology may restrict the amount of data collected for inference. Most current methodologies usually assume that performance characteristics on each link are statistically independent of all other links. However, this assumption can be easily violated due to common cross-traffic flowing through the links. Temporal stationarity is also assumed in many cases. When inference is from end-to-end delay measurements, it is generally assumed that synchronized clocks are available at all sender and receiver nodes. Although these simplified assumptions may not strictly hold, such "first-order" approximations have been shown to be reasonable for the large-scale inference problems [14]. Thus, by the above assumptions, the multicast tree and its internal delay and link loss measurements can be modeled as follows:

- *Tree model.* The physical multicast tree is represented by a tree comprising actual network elements (the nodes) and communication links connecting them. Let $T = (V, L)$ denote a multicast tree with node set V and link set L. The root node 0 is the source of the probe packets, and $R \subset V$ denotes the set of leaf nodes representing the receivers. A link is said to be internal if neither of its endpoints is the root or a leaf node. Let W denote $V \backslash (\{0, 1\} \cup R)$, where 1 is the child node of 0. Each nonleaf node k has a set of children nodes $d(k) = \{d_i(k) \mid 1 \leq i \leq n_k\}$, and each nonroot node k has a parent $p(k)$. The link $\big(p(k), k\big) \in L$ is denoted by link k. We use $j \prec k$ if j is descendant from k, $k = p^r(j)$ if j is an r-level descendant from k, where r is a positive integer. Let $a(U)$ denote the nearest common ancestor of a node set $U \subset V$. Nodes in U are said to be siblings if they have the same parent, that is, if $p(k) = a(U), \forall k \in U$. The subtree of T rooted at k is denoted by $T(k) = \big(V(k), L(k)\big)$, where $V(k)$ and $L(k)$ are node set and link set rooted at k, respectively. The receiver set $R(k)$ is defined as the set of receivers descendant from k, that is, $R(k) = R \cap V(k)$. Figure 69.1 shows an example of multicast tree model.

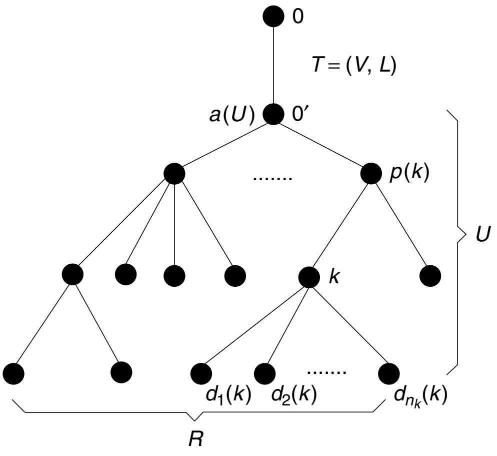


FIGURE 69.1 An example of a multicast tree.

- *Loss/delay model.* Probe packets are dispatched down the tree from the root node 0. Each packet arriving at a node k gives rise to a copy sent to each child node of k. On each link, the packet is either lost, or transmitted with some delay. The delay can be represented as the sum of a fixed propagation delay and a variable queueing delay. Suppose Z_k is the random variable that specifies the queueing delay of a packet traversing link k, $Z_k \in [0, \infty]$. $Z_k = \infty$ denotes packet loss. By convention $Z_0 = 0$. The queueing delay for the path from the root to a node k is $Y(k) = \sum_{j \geq k} Z_k$. If a packet is lost on some link between node 0 and k, $Y(k) = \infty$. Likewise, if a packet does not encounter any queueing delay on each link between node 0 and k, $Y(k) = 0$.

  Assume $\alpha_l(k)$ is the probability of a successful transmission on link k, and $\alpha_u(k)$ the probability of transmission without queueing delay on link k, that is, $\alpha_l(k) = P[Z_k < \infty]$ $\alpha_u(k) = P[Z_k = 0]$. Thus, $1 - \alpha_l(k)$ denotes the probability of a packet lost on link k. $1 - \alpha_u(k)$ denotes the probability of link k being utilized because $Z_k > 0$ iff the link is utilized, which is also called link utilization. If $0 < \alpha_{l/u}(k) < 1, \forall k \in V\backslash\{0\}$, the loss tree is said to be a canonical tree. Any tree (T, α) in noncanonical form can be reduced to a canonical tree [7] by simply removing all subtrees rooted at the broken links whose successful transmission rates are 0. Henceforth, only canonical loss trees are considered in this chapter. A loss tree can thus be modeled as (T, α_l). Similarly, a delay tree can be modeled as (T, α_u).

  For each link an independent Bernoulli loss (delay) model is assumed for each probe packet *being successfully transmitted* (or *transmitted without delay*) across link k with probability $\alpha_l(k)$ $(\alpha_u(k))$. Thus Z_k are independent random variables, and the progress of each probe packet down the tree can be described by Markov stochastic processes $X_{l/u} = (X_{l/u}(k))_{k \in V}$ [6]. Here X_l denotes the loss process and X_u the utilization process. For the loss process, $X_l(k)$=1, if the probe packet reaches k (i.e., $Y(k) < \infty$) and 0 otherwise (i.e., $Y(k) = \infty$), $k \in V$. For the utilization process, $X_u(k)$=1, if the probe packet reaches k without queueing delay (i.e., $Y(k) = 0$) and 0 otherwise (i.e., $Y(k) > 0$), $k \in V$. Their Markov properties follow from the following facts:

$$X_{l/u}(0) = 1;\ X_{l/u}(p(k)) = 0 \Longrightarrow X_{l/u}(k) = 0$$
$$P[X_{l/u}(k) = 1 | X_{l/u}(p(k)) = 1] = \alpha_{l/u}(k)$$

  Obviously, the loss and utilization processes are formally identical except that these processes represent the event of "no loss" and "no delay," respectively.

By the above modeling, it is clear that the observed end-to-end measurements are $(X_{l/u}(k))_{k \in R}$. Given $(X_{l/u}(k))_{k \in R}$, the objective is to infer the network topology, the network internal link-level loss and delay performance which have been modeled as $T = (V, L)$ and $(T, \alpha_{l/u})$.

69.3 Hamming Distance-Based Multicast Network Topology Inference

In this section, we first consider topology inference for multicast networks in the form of a binary for simplicity. We begin with discussing the problems with the previous algorithms for multicast topology inference. To overcome the disadvantage of existing approaches, we introduce the Hamming distance approach and the BHC algorithm. Because of its simplicity and efficiency, the Hamming distance approach plays an important role in identifying siblings in the BHC algorithm. The inference accuracy obtained by the Hamming distance approach and the previous estimation methods is compared.

69.3.1 Existing Approach for Siblings Classification

In the previous papers [5–7,10], network topology was inferred by the identification of siblings from end-to-end measurements. Thus, identification of each siblings pair is the key for topology inference. The previous approaches mainly used the notation $A(i, j)$ to determine whether nodes i and j are siblings, or an equivalent notation to $A(i, j)$.

$A(i, j)$ is defined as the probability that a probe packet *reaches the nearest common ancestor of node pair* (i, j) *successfully* (if using loss measurements) or *reaches it without queueing delay* (if using delay measurements). Obviously, when $A(i, j)$ is minimized, nodes i and j are most likely to be siblings than other node pairs. $A(i, j)$ is obtained by the measurements of loss or delay (also called utilization) as we discussed in Section 69.2. Let $X(k)$ generalize the measurements for loss and delay $X_{l/u}(k)$. In what follows, we use X_k instead of $X(k)$ for simplicity. For the mth probe packet we let $X_k^{(m)}$ be 1 if it reaches node k. Otherwise, $X_k^{(m)}$ is set to 0, indicating that node k did not receive any copy of the probe packet m. Similar for those internal routers, $X_k^{(m)}$ is 1 if the mth probe packet reaches it and 0 otherwise. In practice, $X_k^{(m)}$ for an internal node is obtained by $\vee_{l\in R(k)} X_l^{(m)}$, because the internal node is said to receive a probe packet surely if any receiver descendant from it receives the probe packet. Thus, each node has a "0–1" sequence $\{X_k^{(m)}\}$, $0 < m < n$, $k \in V$. Because the paths from the root to the receivers are different, each sequence maintained by any node is, more or less, different from others. We can thus compare the correlation between the sequences so as to reconstruct the multicast network topology and infer the internal link characteristics. For any node pair (i, j), $i \neq j \neq a(i, j)$,

$$A(i, j) = \frac{P[\vee_{k\in R(i)} X_k = 1]\, P[\vee_{k\in R(j)} X_k = 1]}{P[\vee_{k\in R(i)} X_k = \vee_{k\in R(j)} X_k = 1]} \tag{69.1}$$

Because it is impossible to obtain in practice each probability in Eq. (69.1), $A(i, j)$ is estimated by $A^{(n)}(i, j)$ as used in all previous work, where n is the number of probe packets.

$$A^{(n)}(i, j) = \frac{\sum_{m=1}^{n} X_i^{(m)} \sum_{m=1}^{n} X_j^{(m)}}{n \sum_{m=1}^{n} X_i^{(m)} X_j^{(m)}} \tag{69.2}$$

where $X_i^{(m)} = \vee_{k\in R(i)} X_k^{(m)}\ X_k^{(m)}$, $m = 1, \ldots, n$ denotes the measured outcomes observed at receiver k by n probe packets. Each probability in Eq. (69.1) is estimated by the observation from n probe packets as in Eq. (69.2), for example, $P[\vee_{k\in R(i)} X_k = 1]$ is estimated by $\frac{\sum_{m=1}^{n} X_i^{(m)}}{n}$. As n goes to infinity, $A^{(n)}(i, j)$ is consistent with $A(i, j)$, $i, j \in V$. Then the key clue of multicast network topology inference is minimizing $A^{(n)}(i, j)$ in each iteration and identifying the node pair (i, j) as siblings. For simplicity in describing siblings identification by this estimation approach, we call it the *A-approach* in later sections.

It is worth noting that n is always a finite number of probe packets. Because large numbers of probe packets decrease efficiency, the number of probe packets is set to be small in many cases. Thus, each probability in Eq. (69.2) is not consistent to the real probability. The estimation for $A(i, j)$ may cause obvious mistakes as shown in Fig. 69.2, when determining whether two nodes are siblings with a finite number of probe packets.

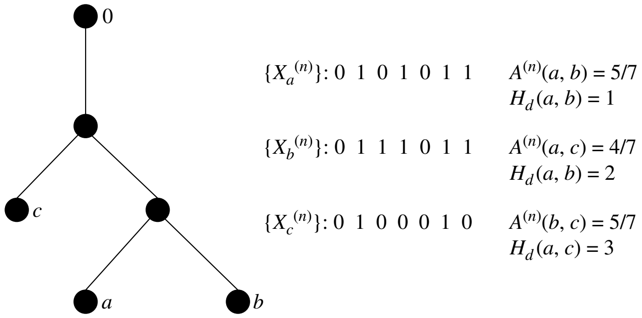


FIGURE 69.2 Comparison of Hamming distance($H_d(\cdot, \cdot)$) and the previous A-approach ($A^{(n)}(\cdot, \cdot)$) for each pair node.

According to the estimated value $A^{(n)}(i, j)$ of Eq. (69.2), if receiver j loses many probe packets, the value of $A^{(n)}(i, j)$ will be very small, even smaller than the value between i and its actual sibling, for example, $A^{(n)}(a, c) < A^{(n)}(a, b)$, $n = 7$ in Figure 69.2. Thus, the A-approach will misclassify nodes a and c as siblings. Even when the number of probe packet increases, such bias cannot be eliminated completely. Because the estimated probability in Eq. (69.2) does not equal to the true probability unless the number of probe packets is infinite. To reduce the bias with a finite number of probe packets, we use the Hamming distance classification approach.

69.3.2 Hamming Distance Classification Approach

The Hamming distance between nodes u and v is defined as the number of different bits between their sequences, which is given in the following equation, where "$\oplus$" is the exclusive-OR operator:

$$H_d(u, v) = \sum_{m=1}^{n} X_u^{(m)} \oplus X_v^{(m)} \tag{69.3}$$

For instance, the Hamming distances between each pair of sequences are given in Figure 69.2. Different Hamming distances between different node pairs can be used to determine which pair of nodes are siblings. As illustrated in Figure 69.2, $H_d(a, c) > H_d(a, b)$ is congruent with the relationship between the nodes because nodes a and b are siblings. This is to say, the Hamming distance approach distinguishes the siblings and nonsiblings with different values successfully. However, the A-approach failed to identify the relationship of nodes a, b, and c with seven probe packets in Figure 69.2, because $A^{(n)}(i, j)$ shows the contrary relationship between node pairs (a, c) and (a, b). We will discuss the reason why our Hamming distance approach is superior to the A-approach in Section 69.3.4.

In the multicast network, the nearer the two nodes are located, the more similar the two "0–1" sequences they maintained are. The reason for such phenomenon is that the probe packets from the root to the receivers may pass many common links. If the receivers are siblings, the paths the probe packets pass from the root to their parent nodes are the same. Therefore, the correlation of two "0–1" sequences between a node and its siblings is greater than that of all other pairs of sequences between a node and one of its nonsibling nodes.

To infer the topology of the multicast network, all the siblings need to be identified correctly. That is, we should find out the similarity between each pair of sequences. The problem of identifying siblings in the multicast network can be stated as marking out the similarity and dissimilarity of all pairs of bit sequences. For a two-component sequence, Hamming distance is the simplest and most efficient method to identify the similarity and dissimilarity among different sequences. Therefore, we use Hamming distance to identify siblings for multicast network topology inference.

It should be noted that the Hamming distance classification approach generally works well regardless of temporal correlation between the losses in actual networks. No matter when we send the probe packets and how many probe packets we use to infer the network topology, the Hamming distance approach can classify siblings correctly. In contrast, the previous A-approach is affected strongly by the temporal correlation of collected data. Different data collected in different time periods may cause different siblings classifications, especially when a small number of probe packets is used.

69.3.3 Binary Hamming Distance Classification Algorithm

The BHC algorithm is based on the Hamming distance classification approach. BHC also incorporates the hop count measurements as HBLT (Binary Loss Tree classification with Hop count) proposed in Ref. [10]. Each node k is associated with a hop count $k.hop$. The $k.hop$ values of leaf nodes can be obtained by simply reading the TTL values of the probe packets. For internal nodes, their $k.hop$ values can be computed from the $k.hop$ values of leaf nodes during the topology inference procedure.

In BHC, the nodes with the same value of hop count, a node pair, is identified as siblings if the Hamming distance is not only minimal among all Hamming distances of all node pairs in the node set, but also less than a given threshold. The BHC algorithm is run in a bottom-up fashion, which is described in detail below.

```
Input: The set of receivers $R$, number of probe packets $n$, observed sequences at
  receivers $(X_k^{(i)})_{k\in R}^{i=1,\ldots,n}$;
$R' := R$, $V' := \emptyset$, $L' := \emptyset$, $h = \max_{k\in R}(k.hop)$, $W_e = \emptyset$, $(e = 1, \ldots, h)$; //$V'$ is the set
  of discovered nodes; $L'$ is the set of discovered links, $W_e$ is a set of nodes with hop count
  value $e$, $e$ is initialized as the maximum value of hop count for all nodes in $R$.//
for $k \in R$, do
    $W_{k.hop} := W_{k.hop} \bigcup \{k\}$ //Classify the receivers into different groups according to their
    hop count values.//
while $e > 1$ do
    while $W_e \neq \emptyset$ do
        Let $u$ be the first element in $W_e$; search for $v \in W_e$ to minimize $H_d(u, v)$, $(u \neq v)$;
        if $H_d(u, v) > \delta_e$ then $S = \{u\}$, Set $r$ to be $u$'s virtual parent node; //Initially, sibling
          nodes set $S := \emptyset$, $\delta_e$ is a given classification threshold of level $e$. If the minimal
          Hamming distance is still greater than $\delta_e$, $u$ does not have siblings.//
        else $S = \{u, v\}$, Set $r$ to be $u$ and $v$'s virtual parent node; //$u$ and $v$ are siblings,
          $r$ is denoted as their virtual parent node.//
        for $i = 1, \ldots, n$ do $X_r^{(i)} := \vee_{l\in S} X_l^{(i)}$;
        $r.hop := e - 1$;
        $V' := V' \bigcup S$; $W_e := W_e \setminus S$; $W_{e-1} := W_{e-1} \bigcup \{r\}$; //The discovered two siblings
          are added to the discovered node set $V'$ and excluded from original set $W_e$, their
          parent node is added to the node set $W_{e-1}$.//
        for each $l \in S$, $L' := L' \bigcup \{(r, l)\}$;
        $S = \emptyset$;
    $e := e - 1$;
$V' := V' \bigcup \{0\}$; $L' := L' \bigcup \{0, r\}$; //Include root node and the link from root node to his child
  node to the discovered node set and link set.//
Output: Inferred topology $(V', L')$.
```

Thus, the operation of BHC can be described as follows. First, all the receivers are classified into different node sets W_e $(1 \leq e \leq h)$ according to their values of hop count. Inference begins by identifying siblings in the node set with the maximum value of hop count. The Hamming distances of each node pair in W_e are calculated. The node pair is identified to be siblings if its Hamming distance is minimal and less than the threshold. Remove the siblings from the node set with the hop count being e and add the parent node into the node set with hop count reduced by 1. The "0–1" sequence of the parent node is obtained by the "OR" operation of those of the siblings. When all nodes in W_e are grouped decrease hop count value by 1. Repeat the same procedure among the nodes in the node set W_{e-1}. The algorithm ends when the hop count becomes 1.

Therefore, by the Hamming distance classification approach, the network topology can be efficiently inferred from multicast end-to-end measurements on loss or delay of probe packets.

69.3.4 Analysis on Inference Accuracy of Hamming Distance Classification Approach

The Hamming distance-based topology inference algorithm, that is, BHC in this chapter and the previous A-approach-based topology inference algorithms can all find a consistent result with the real multicast network as the number of probe packets increases to infinity. However, with a finite number of probe packets, the BHC algorithm can obtain accurate results with a higher probability than previous algorithms, because the Hamming distance approach has been found to be superior to the A-approach used in Refs. [5–7,10] for siblings classification.

Definition 69.1

Let s_1 and s_2 be two nodes that have a common parent node i, s_3 be a node for which node i is not its parent, W_k is the node set with the hop count k. Define $s(i)$ as follows:

$$s(i) = \{(s_1, s_2, s_3) : \forall s_1, s_2, s_3 \in W_k, 1 \le k \le h\}$$

Definition 69.2

For $(s_1, s_2, s_3) \in s(i)$, let $D_H(s_1, s_2, s_3)$ be the difference between the Hamming distance of non-siblings and siblings, and $D_A(s_1, s_2, s_3)$ be the difference between $A^{(n)}(\cdot, \cdot)$ of nonsiblings and siblings. That is,

$$\begin{aligned} D_H(s_1, s_2, s_3) &= H_d(s_1, s_3) - H_d(s_1, s_2) \\ D_A(s_1, s_2, s_3) &= A^{(n)}(s_1, s_3) - A^{(n)}(s_1, s_2) \end{aligned}$$

Lemma 69.1

Sufficient conditions for correctly identifying nodes s_1 and s_2 as siblings are $0 < min_{(s_1,s_2,s_3)\in s(i)} D_H(s_1, s_2, s_3)$ and $H(s_1, s_2) < \delta_e$. For the A-approach, the same condition must be satisfied by replacing $D_H(s_1, s_2, s_3)$ with $D_A(s_1, s_2, s_3)$, and $H(s_1, s_2)$ with $A^{(n)}(s_1, s_2)$.

Lemma 69.1 holds because the Hamming distance or $A^{(n)}(\cdot, \cdot)$ of a node and its nonsibling nodes should be greater than that of it and its siblings.

We denote by $n^1_{s_i}$ the number of probe packets transmitted from the root to node s_i successfully, and by $n^1_{s_i s_j}$ the number of probe packets transmitted successfully from the root to both nodes s_i and s_j at the same time, $i = 1$–3.

Lemma 69.2

For $(s_1, s_2, s_3) \in s(i)$, if inequality (69.4) holds, the Hamming distance approach can identify the siblings while $A^{(n)}(\cdot, \cdot)$ cannot; if inequality (69.5) holds, $A^{(n)}(\cdot, \cdot)$ can identify the siblings while the Hamming distance approach cannot; in all other cases, both approaches can identify siblings correctly.

$$\frac{1}{2}\left(n^1_{s_2} - n^1_{s_3}\right) < n^1_{s_1 s_2} - n^1_{s_1 s_3} \le \frac{n^1_{s_1 s_3}}{n^1_{s_3}}\left(n^1_{s_2} - n^1_{s_3}\right) \tag{69.4}$$

$$\frac{n^1_{s_1 s_3}}{n^1_{s_3}}\left(n^1_{s_2} - n^1_{s_3}\right) < n^1_{s_1 s_2} - n^1_{s_1 s_3} \le \frac{1}{2}\left(n^1_{s_2} - n^1_{s_3}\right) \tag{69.5}$$

Proof

Since $H_d(s_1, s_2) = n^1_{s_1} + n^1_{s_2} - 2n^1_{s_1 s_2}$ and $A^{(n)}(s_1, s_2) = \frac{n^1_{s_1} \cdot n^1_{s_2}}{n \cdot n^1_{s_1 s_2}}$, we have,

$$\begin{aligned} D_H(s_1, s_2, s_3) &= n^1_{s_1} + n^1_{s_3} - 2n^1_{s_1 s_3} - \left(n^1_{s_1} + n^1_{s_2} - 2n^1_{s_1 s_2}\right) \\ &= \left(n^1_{s_3} + 2\left(n^1_{s_1 s_2} - n^1_{s_1 s_3}\right)\right) - n^1_{s_2} \end{aligned} \tag{69.6}$$

and

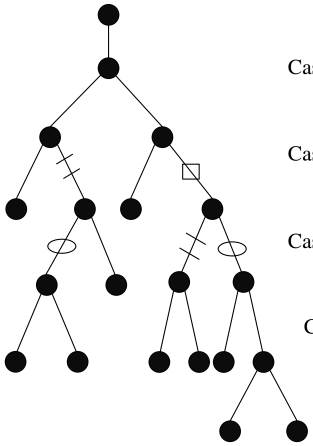
$$D_A(s_1, s_2, s_3) = \frac{n^1_{s_1} \cdot n^1_{s_3}}{n \cdot n^1_{s_1 s_3}} - \frac{n^1_{s_1} \cdot n^1_{s_2}}{n \cdot n^1_{s_1 s_2}} \tag{69.7}$$
$$= \frac{n^1_{s_1}}{n \cdot n^1_{s_1 s_2}} \cdot \left(\left(n^1_{s_3} + \frac{n^1_{s_3}}{n^1_{s_1 s_3}} \cdot \left(n^1_{s_1 s_2} - n^1_{s_1 s_3} \right) \right) - n^1_{s_2} \right)$$

From Lemma 69.1, we know that nodes s_1 and s_2 will be identified as siblings if $D_H(s_1, s_2, s_3) > 0$ using the Hamming distance approach for any $(s_1, s_2, s_3) \in s(i)$. If (s_1, s_2, s_3) results in $D_H(s_1, s_2, s_3) < 0$, the Hamming distance approach will fail to identify s_1 and s_2 as siblings correctly. Similar conditions holds for the $A^{(n)}(\cdot, \cdot)$ approach. Therefore, we can conclude that if any $(s_1, s_2, s_3) \in s(i)$ results in $D_A(s_1, s_2, s_3) < 0$ while $D_H(s_1, s_2, s_3) > 0$, the Hamming distance approach is superior to the $A^{(n)}(\cdot, \cdot)$ approach in siblings identification, and vice versa. Lemma 69.2 describes the complete conditions based on Eq. (69.6) and Eq. (69.7). □

Lemma 69.2 shows when the Hamming distance approach or the A-approach can identify nodes s_1 and s_2 as siblings correctly. If the probability that inequality (69.4) holds is greater than the probability that inequality (69.5) holds, the Hamming distance approach works better than the previous A-approach. However, obtaining the exact probabilities for inequalities (69.4) and (69.5) to hold are very difficult, because both probabilities vary with different network connections and conditions. Thus, we only analyze the cases when inequalities (69.4) and (69.5) hold, respectively, and give an example through which we can have a clear view on which approach has a better performance in inference accuracy.

First we should note that in most cases, both the Hamming distance approach and the A-approach can identify siblings correctly according to our analysis of the different receiving cases for nodes s_1, s_2, and s_3. When links s_1 and s_2 are in similar conditions, the Hamming distance approach can identify siblings correctly if the A-approach can do so. However, we also find that sometimes the A-approach cannot identify siblings correctly while the Hamming distance approach can. When both nodes s_1 and s_2 receive almost all probe packets while node s_3 loses many probe packets, and $n^1_{s_1 s_3}$ happens to be equal to $n^1_{s_3}$, the Hamming distance approach can identify siblings correctly while the A-approach cannot. The above is also true when both nodes s_1 and s_2 lose many probe packets while s_3 receives almost all probe packets, and $\frac{n^1_{s_1 s_3}}{n^1_{s_3}} < \frac{1}{2}$. In very few cases, the Hamming distance approach may not identify siblings correctly while the A-approach can. This might happen only when links s_1 and s_2 are in different conditions which result in dissimilar sequences on nodes s_1 and s_2. In a multicast network, only a few siblings links may exhibit completely different performances among all siblings–links pairs. Even if under this condition, it can be noted that in most cases the Hamming distance approach can identify siblings correctly as the A-approach does. Thus from all cases discussed, we can see that the occurrence of the Hamming distance approach outperforming the A-approach is more frequent than that of the opposite.

As to the exact probabilities that inequalities (69.4) and (69.5) hold, we assume a multicast network with 5–27% links losing packets severely, the ratios can represent most cases in real multicast networks. Less than 5% links suffering severe packet losses only occur in some applications. Multicast networks with more than 10% links losing packets often result in unacceptable performance because most group receivers can be affected by not receiving packets accurately. Therefore, the multicast network with 5–27% links losing packets severely as shown in Figure 69.3 chooses reasonable ratios for ill-performing links and represents situations arising in practice. Assume other links experience some loss and congestion to a different degree. For this network the four different cases and the probabilities of inequalities (69.4) and (69.5) holding in each case are calculated and displayed in the figure. We find that the probabilities vary with different link status. As we have discussed before, the probabilities also vary with different network topologies. Figure 69.3 illustrates clearly that the probability that the Hamming distance approach succeeds, but the A-approach fails is greater than the probability in the opposite situation for this network. This conclusion may be extended to any network in general. The simulation of the network given in Section 69.3.5 also supports this conclusion.



Case 1: Links with circle mark lose lots of packets (10.5% links)

Case 2: Links with line mark lose lots of packets (10.5% links)

Case 3: Links with rectangular mark lose lots of packets (5.3% links)

Case 4: All marked links lose lots of packets (26.3% links)

Link status	Prob. of Hd outperforming A	Prob. of A outperforming Hd	Prob. of both working well
Case 1	0.3	0.2	0.5
Case 2	0.4	0.2	0.4
Case 3	0.7	0.1	0.2
Case 4	0.4	0.3	0.3

FIGURE 69.3 Comparison on probabilities that inequality (69.4) or (69.5) holds in a certain multicast network.

Though the probabilities that inequalities (69.4) and (69.5) hold are different as the network condition varies, we have seen from the above analysis that with a finite number of probe packets, the Hamming distance approach is more likely to work out the accurate topology than the A-approach. In other words, due to the greater probability of the Hamming distance approach outperforming the A-approach in siblings identification, we may conclude that the use of the Hamming distance approach in the BHC algorithm can result in a better performance than using the A-approach in inference accuracy in our tested networks.

69.3.5 Experimental Results for Topology Inference

This section validates the BHC algorithm by comparing it with HBLT which is based on the A-approach. Because both algorithms take hop count into consideration, the comparison results will clearly show the difference of topology inference by the Hamming distance classification approach and the A-approach.

We executed both algorithms for the network topology shown in Figure 69.4 via the network simulator ns. Node 0 is the sender and nodes 1–10 are the receivers. All internal links are configured with different capacities ranging from 0.1 to 5 Mbit/s. The link $0 \rightarrow 0'$ is set at 5 Mbit/s. The root node 0 generates probe packets in a 20 K bit to 2 Mbit/s stream. Every probe packet comprises one UDP packet with 1000 bytes.

Due to the low capacity and heavy traffic load of links, multicast probe packets can be delayed or even lost. The loss measurements are counted for topology inference. Then for each receiver, the collected data is set to 1 if the probe packet is received, otherwise 0. Both algorithms work on the collected "0–1" sequences. From TTL field of the probe packets obtained from receivers, the hop count required by BHC and HBLT can easily be obtained.

To compare the inferred topologies by the different algorithms, a similarity degree, which is defined below, is used [10].

Definition 69.3

Define SimilarityDegree $= \alpha \cdot s + \beta \cdot h$, *where* s *denotes the ratio of the number of nodes whose siblings are identified correctly to the total number of nodes,* h *denotes the ratio of the number of nodes whose hop levels are inferred accurately to the total number of nodes.* α, β *are the weight of these two factors. When at least one*

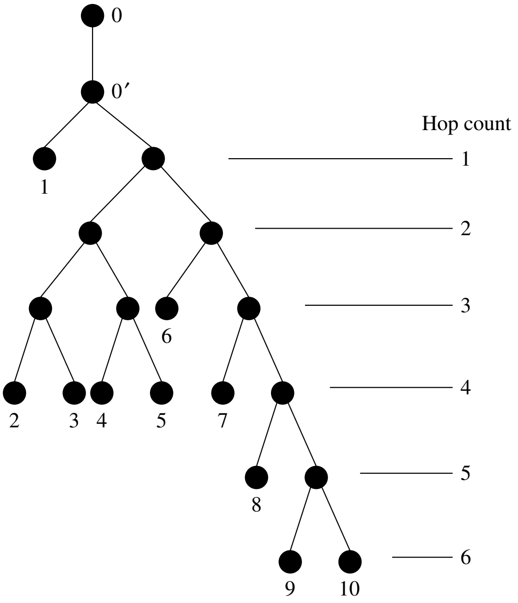


FIGURE 69.4 A multicast network topology.

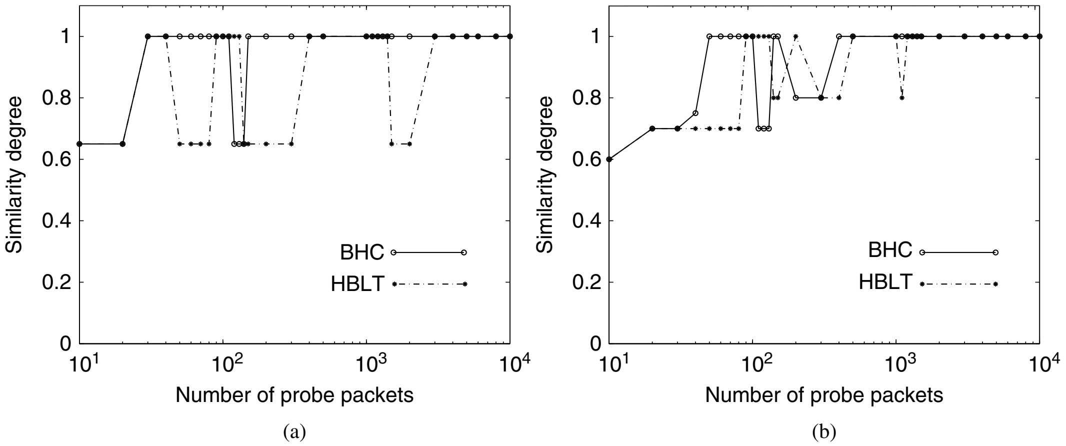


FIGURE 69.5 Similarity degree comparison of HBLT and BHC when $\alpha = 0.5$, $\beta = 0.5$.

of the two subtrees of a node's sibling is the same as that of the physical tree, we say that the node is inferred correctly.

We let $\alpha = 0.5$ and $\beta = 0.5$, then the compared results are shown in Figure 69.5 [11].

Figure 69.5(a) and Figure 69.5(b) are obtained by changing different links' capacities. Both simulation results show that BHC requires fewer probe packets than HBLT to infer the accurate topology constantly. Therefore, we can conclude that BHC is more efficient in topology inference than the HBLT algorithm.

Figure 69.5(a) and Figure 69.5(b) also validate our analysis on siblings identification by the Hamming distance approach and the A-approach. It shows that the occurrence of the Hamming distance approach outperforming the A-approach is more frequent than that of the opposite situation. This supports the conclusion we have drawn in Section 69.3.4, that is, the probability that Hamming distance approach outperforming the A-approach should be greater than the probability of the opposite situation.

69.4 Hamming Distance Matrix for Internal Loss/Delay Performance Analysis

From the previous section, it is seen that every receiver maintains a bit sequence from end-to-end measurements. On the basis of these measurements and the discovered topology, we present a Hamming distance matrix-based scheme for analyzing the network link loss and internal delay performance.

We begin by defining a Hamming distance matrix. The bit sequence maintained by each receiver is denoted by $\{X_r^{(m)}\}$, $1 \leq m \leq n$, $r \in R$. We assume the number of receivers in a multicast network to be l. For simplicity, let d_{ij} denote the Hamming distance between receiver i and j, that is, $d_{ij} = H_d(i, j)$. Denote D to be the Hamming distance matrix. Then D is defined as follows:

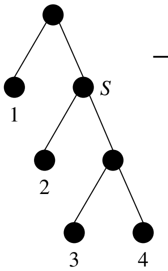
$$D = \begin{pmatrix} d_{11} & d_{12} & \dots & d_{1l} \\ d_{21} & d_{22} & \dots & d_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ d_{l1} & d_{l2} & \dots & d_{ll} \end{pmatrix}.$$

Obviously, the matrix is symmetric, that is, $d_{ij} = d_{ji}$ for $i \neq j$. And it is also easy to see that $d_{ij} = 0$ if $i = j$. Apart from these properties, the matrix supplies additional information. For instance, if receiver i and j are siblings, d_{ij} is supposed to be smaller than d_{ik}, $k \neq j$ and $k \neq i$. More generally, the closer the two nodes are located, the smaller their Hamming distance is supposed to be because their sequences are more similar due to more shared common link condition which we have discussed in Section 69.3. Therefore, such a Hamming distance matrix provides a lot of information of the internal topology when the internal links have different congestion and loss rates.

Now let us see what this matrix can do for link loss and internal delay performance analysis and identification. Consider the network in Figure 69.6 where only one receiver observes severe loss. We assume here that the loss measurements are counted. Internal link delay performance can be obtained if the end-to-end delay measurements observed on receivers are counted. Suppose link s in Figure 69.6 is in very poor condition and all other links are in good condition and thus have few losses.

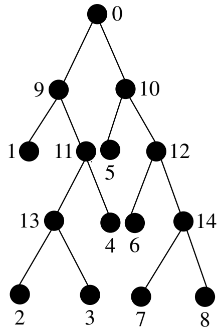
According to the Hamming distance matrix in Figure 69.6, we can easily find that the Hamming distances between 1 and any of the remaining receivers are very large, while the Hamming distances among receivers 2, 3, and 4 are quite similar and small. Then we can infer that the nearest common link of receivers 2, 3, and 4 work in poor condition.

As for a network with very complex internal link conditions, we need to transform the matrix into several blocks according to the different values of components in the matrix. Usually we classify the components by the experienced difference which depends on how many probe packets are sent in total. With the experienced difference, we can do elementary transformations on the Hamming distance matrix



$$\begin{array}{c|cccc} \text{Receiver} & 1 & 2 & 3 & 4 \\ \hline D = & \multicolumn{4}{c}{\begin{pmatrix} 0 & 88 & 85 & 92 \\ 88 & 0 & 0 & 3 \\ 85 & 0 & 0 & 2 \\ 92 & 3 & 2 & 0 \end{pmatrix}} \end{array}$$

FIGURE 69.6 A simple Hamming distance matrix.



$$D = \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline \end{array}$$

$$D = \begin{pmatrix} 0 & 72 & 73 & 0 & 0 & 91 & 102 & 102 \\ 72 & 0 & 3 & 73 & 75 & 43 & 52 & 51 \\ 73 & 3 & 0 & 61 & 62 & 53 & 66 & 63 \\ 0 & 73 & 61 & 0 & 0 & 92 & 105 & 101 \\ 0 & 75 & 62 & 0 & 0 & 96 & 101 & 102 \\ 91 & 43 & 53 & 92 & 96 & 0 & 27 & 30 \\ 102 & 52 & 66 & 105 & 101 & 27 & 0 & 0 \\ 102 & 51 & 63 & 101 & 102 & 30 & 0 & 0 \end{pmatrix}$$

$$\begin{array}{ccc:cc:c:cc} 1 & 4 & 5 & 2 & 3 & 6 & 7 & 8 \\ \hline \end{array}$$

$$\left(\begin{array}{ccc:cc:c:cc} 0 & 0 & 0 & 72 & 73 & 91 & 102 & 102 \\ 0 & 0 & 0 & 73 & 61 & 92 & 105 & 101 \\ 0 & 0 & 0 & 75 & 62 & 96 & 101 & 102 \\ \hdashline 72 & 73 & 75 & 0 & 3 & 43 & 52 & 51 \\ 73 & 61 & 62 & 3 & 0 & 53 & 66 & 63 \\ \hdashline 91 & 92 & 96 & 43 & 53 & 0 & 27 & 30 \\ \hdashline 102 & 105 & 101 & 52 & 66 & 27 & 0 & 0 \\ 102 & 101 & 102 & 51 & 63 & 30 & 0 & 0 \end{array}\right)$$

Hamming distance matrix in blocks

FIGURE 69.7 Hamming distance matrix and matrix in blocks.

and obtain a matrix with several regular blocks. The receivers are classified into several groups. Thus we can infer all the possible bad links in the topology according to those classified receivers by blocks easily. For example, the network topology, the Hamming distance matrix, and the transformed matrix are given in Figure 69.7; we aim to analyze the internal delay and link loss performance and locate the ill-performing link easily.

There is a group including receivers 1, 4, and 5 which receive almost all the probe packets. This group is called base group which can be easily identified by the 0–1 sequence maintained by any receiver in this group. The links in the path from the source to the receivers in this group all work in good condition. We find all other groups have a common property, that is, the Hamming distance between any receiver in these groups and any receiver in base group is very large while the Hamming distance between the receivers in each group is very small. These groups are called loss group (or delay group in the case of internal delay inference). We can then decide that the link connecting to the nearest common ancestor node of each loss/delay group is one of the links which is *causing a severe loss* (experiencing severe delay). In the topology of Figure 69.7, links 13, 6, and 14 are the links we aim to identify. All other links work well with few loss or delay in this network.

Therefore, if we can transform the Hamming distance matrix into blocks according to the value of the components, the receivers are classified into several groups. Then we can determine a base group which include those receivers who *receive almost all probe packets* (or *receive most probe packets without delay*), that is, there is not a any link causing big loss (or severe delay) in the path from the source to these receivers. And apart from this group, there are many loss/delay groups. The links connecting to the nearest common ancestor node of each loss/delay group are identified as those links where severe loss/delay exists. This means, the Hamming distance matrix D of a network can help to analyze and identify the network internal loss/delay performance. Moreover, a well-known phenomenon is that there are usually only a few links that are in poor condition in a large-scale network in practice. Therefore, the components in the Hamming distance matrix usually appear to be different obviously and can be easily classified.

69.5 Topology Inference for General Trees

This section extends the Hamming distance classification approach for topology inference to general trees.

It is more complicated to infer the topology for general trees than for binary ones. We thus introduce a threshold ϵ into grouping the siblings set S. The set S is grouped if the Hamming distance between any pair of nodes in S is sufficiently close to being minimal.

The grouping step starts by finding a pair of nodes $\{u, v\}$ that has the minimum Hamming distance in S, then adjoining further elements to it provided the following inequality is satisfied:

$$H_d(u, v')(1 - \epsilon) < H_d(u, v) \tag{69.8}$$

Thus we replace line 9 of the BHC algorithm by the following steps so that topology inference for general trees can be performed.

```
9a. else { $S = \{u, v\}$;
9b.     while there exists $v' \in W_e \backslash S$ such that $H_d(u, v')(1 - \epsilon) < H_d(u, v)$ do
9c.     $S := S \cup \{v'\}$; }
9d. Set $r$ to be the virtual parent node of all identified siblings in $S$.
```

We also use classification threshold δ_e for identification of siblings in general trees in a similar way as it was used in the BHC algorithm of Section 69.3.3. We can set δ_e to a given experience value for a network with binary tree topology due to the structural simplicity. However, for general trees, we set δ_e to be $\frac{n}{k} \log e$ to reduce the ratio of misclassification of siblings. Here e is the level of nodes computed from the root, n the total number of nodes in the multicast network, and k the estimated expected number of branches of the multicast network. We set δ_e to be $\frac{n}{k} \log e$ because we want δ_e to be linearly proportional to the number of nodes and to the logarithm of the level, and inversely proportional to the branches of each level in the multicast tree. The value may also need to be adjusted according to the real condition of the network.

As pointed out in Ref. [7], the violation of the condition described in inequality (69.8) has the interpretation that the ancestor $a(U)$ is separated from $a(\{u, v\})$ by a link with loss rate at least ϵ. The convergence of the inferred topology to the true topology is mainly influenced by ϵ. If only ϵ is less than the internal link loss rates, the inferred topology will be convergent to the true topology. However, the internal link loss rates are unknown in advance. A small value of ϵ is more likely to satisfy the above condition but at the cost of slow convergence. A large value of ϵ, in contrast, is more likely to result in systematically removing links with small loss rates. Thus it is convergent to a wrong topology. To choose an appropriate ϵ to obtain more accurate and complete topology practically, the loss rate inference is extremely helpful for general topology inference for which we have proposed the scheme of incorporating the link loss rate into topology inference in Ref. [15].

69.6 Concluding Remarks

We introduced the Hamming distance approach for multicast topology discovery and network internal characteristics inference in this chapter. This approach was proposed in our previous work and contents of its applications are collected from our recent publications [9–11]. We showed that Hamming distance of sequences obtained from multicast end-to-end loss/delay measurements for each pair of nodes can be used effectively, not only in network topology inference, but also in network link loss and internal delay performance inference.

References

[1] Levine, B. N., Paul, S., and Garcia Luna Aceves, J. J., Organizing multicast receivers deterministically according to packet-loss correlation, *Proc. of ACM Multimedia*, 1998.

[2] Li, D. and Cheriton, D. R., Oters (on-tree efficient recovery using subcasting): a reliable multicast protocol, *Proc. of IEEE Int. Conf. on Network Protocols*, 1998, p. 237.

[3] Castro, R., Coates, M., and Nowak, R., Maximum likelihood identification from end-to-end measurements, *Proc. of DIMACS Workshop on Internet Measurement, Mapping and Modeling*, DIMACS Center, Rutgers University, 2002.

[4] Caceres, R., Duffield, N. G., Horowitz, J., Lo Presti, F., and Towsley, D., Loss based inference of multicast network topology, *Proc. of IEEE Conf. on Decision and Control*, 1999.

[5] Duffield, N. G., Horowitz, J., Lo Presti, F., and Towsley, D., Multicast topology inference from end-to-end measurements, *ITC Seminar on IP Traffic Measurement, Modeling and Management*, 2000.

[6] Duffield, N. G., Horowitz, J., and Lo Presti, F., Adaptive multicast topology inference, *Proc. of INFOCOM*, 2001, p. 1636.

[7] Duffield, N. G., Horowitz, J., Lo Presti, F., and Towsley, D., Multicast topology inference from measured end-to-end loss, *IEEE Trans. Inf. Theor.*, 48(1), 26, 2002.

[8] Ratanasamy, S. and McCanne, S., Inference of multicast routing tree topologies and bottleneck bandwidths using end-to-end measurements, *Proc. of INFOCOM*, 1999, p. 353.

[9] Tian, H. and Shen, H., Discover multicast network internal characteristics based on Hamming distance, *Proc. IEEE ICC*, 2005.

[10] Tian, H. and Shen, H., An improved algorithm of multicast topology inference from end-to-end measurements, *Int. J. Commun. Syst.*, 2005.

[11] Tian, H. and Shen, H., Hamming distance and hop count based multicast network topology inference, *Proc. of IEEE AINA*, 2005, p. 267.

[12] Caceres, R., Duffield, N. G., Horowitz, J., and Towsley, D., Multicast-based inference of network-internal loss characteristics, *IEEE Trans. Inf. Theor.*, 45(7), 2462, 1999.

[13] Duffield, N. G. and Lo Presti, F., Multicast inference of packet delay variance at interior network links, *Proc. of IEEE INFOCOM*, 1351, 2000, p. 1351.

[14] Coates, M. J., Hero, A. O., Nowak, R., and Yu, B., Internet tomography, *IEEE Signal Process. Mag.*, 19(3), 47, 2002.

[15] Tian, H. and Shen, H., Multicast Based Inference for Topology and Network-Internal Loss Performance from End-to-end Measurements, Computer Communications, Elsevier, 29(11), 1936, 2006.

70 Multicast Congestion in Ring Networks

SingLing Lee
National Chung-Cheng University

RongJou Yang
Wufeng Institute of Technology

Hann-Jang Ho
Wufeng Institute of Technology

70.1 Introduction

Due to the recently rapid development of multimedia applications, multicast has become the critical technique in many network applications. In multicasting routing, the main objective is to send data from one or more sources to multiple destinations in order to minimize the usage of resources such as bandwidth, communication time, and connection costs. We investigate contemporary research concerning multicast congestion problems. These problems include multicast Steiner tree, multicast packing, undirected Minimum Congestion Hypergraph Embedding in a Circle (MCHEC), and directed MCHEC problem. We discuss randomized metarounding for these problems. The multicast congestion problem is to find a set of multicast trees that minimize the maximum congestion over all its edges. The congestion of an edge is the number of multicast trees that use the edge. The topology of multicast congestion research can be classified into two categories: general graphs and ring networks, we focus on multicast congestion on ring networks and discuss the undirected and directed version in detail in Sections 70.2 and 70.3, respectively.

For general graphs, given a physical network $G = (V, E)$ with a set V of n nodes, a set E of undirected network links and m multicast requests $S = S_1, S_2, \ldots, S_m$ being subsets of V, a solution to the problem is a set of m trees such that the ith tree spans the nodes of the ith multicast request. The objective function is to minimize the maximum congestion. The problem was formulated as an Integer Linear Programming (ILP) and its LP relaxation solution finds fractional solutions for each multicast request. Vempala and Vöcking [1] proposed an algorithm with approximation bound of $O(\log n)$ based on an LP relaxation where the fractional solution for each multicast is a set of paths rather than a set of trees with fractional weights. Raghavan and Tompson [2] presented an exponential (with respect to number of nodes in the multicast tree, but polynomial with respect to the number of different multicasts) time algorithm that generates solutions with congestion $O(OPT + \log n)$, where OPT denotes the optimal solution value.

Carr and Vempala [3] proposed a polynomial-time algorithm with the same quality based on the ellipsoid method by decomposing the fractional solution for each multicast into a combination of trees. They then employ a simple rounding algorithm by decomposing the fractional solution into a convex combination of trees and by selecting one tree with probability equal to its convex multiplier to find its fractional congestion on an edge. If the fractional solution for each multicast can be formulated as a convex combination of trees, then the congestion on an edge will be exactly its fractional congestion. Furthermore, this decomposing process is independent for each multicast, thus the deviation from the expectation was bounded by a solution within $c \cdot OPT + O(\log\ n)$, for some constant c. The solutions are Steiner trees spanning the vertices of the multicast.

In the multicast packing problem, the network tries to accommodate simultaneously all the multicast groups (many-to-many) and avoid bottlenecks on the links to achieve higher throughput (i.e., minimize the maximum link sharing among the multicast groups). A shared tree can be considered as the backbone of a group multicasting session. One way to minimize the maximum congestion is to increase the size of some multicast trees, but this also increases the delay which must be considered in the objective function of the optimal packing problem formulation. The delay is a function of the amount of dilation α from the size of the optimal tree obtained for each group multicast independently from the others (i.e., in isolation). In Ref. [4], Chen, G*ü*nl*ü*k, and Yener proposed a suboptimal solution to the Steiner tree problem (which is known to be NP-hard) to reduce the sharing of a link in order to ensure that the size of multicast trees will never exceeds $\alpha \cdot OPT^k$ (OPT^k is the cost of the optimum tree for multicast group k in isolation). The optimal multicast tree for each group in isolation was computed by using cutting-plane inequalities (see Ref. [5] for details) and a branch-and-cut algorithm.

For ring networks, an optimal solution to the Minimum Congestion Graph Embedding in a Cycle problem (undirected MCGEC) for a set of routing requests, where each request is defined by a pair of network nodes (i.e., a source and a destination) to be connected, can be solved in polynomial time based on the graph theoretical approach developed by Okamura and Seymour [6,7]. The weighted undirected MCGEC problem has a polynomial time approximation algorithm proposed in Refs. [8,9]. The undirected MCHEC problem is to embed the hyperedges of a hypergraph as paths in a cycle such that the maximum congestion is minimized. The MCHEC problem can be mapped into the multicast ring network problem by viewing the hyperedges and the hypergraph as the multicast groups and the multicast ring topology, respectively.

Ganley and Cohoon [10] proposed the problem of hypergraph embedding in a cycle. The hypergraph embedding problem is to embed m hyperedges as adjacent paths of an n-vertex cycle such that the maximum number of adjacent paths over any physical link of the cycle is minimized. This problem is NP-complete in general, but solvable in polynomial time when the congestion of embedded paths is at most k. For any fixed k, a solution can be computed in $O((nm)^{k+1})$ time. They also proposed an approximation algorithm that is guaranteed to find a solution with the maximum congestion at most three times that of the optimal solution.

For unknown maximum congestion, Gonzalez [11] proposed two improved approximation algorithms that both generate solutions within two times the optimum. The first algorithm transforms the hypergraph embedding problem to a linear programming (LP) formulation, and the other one solves the problem in linear time by transforming hyperedges to normal edges. This algorithm is discussed in Chapter 3.

Carpenter et al. [12] provides a linear time approximation algorithm which routes the hyperedges in the clockwise direction starting from the lowest numbered vertex to the highest numbered vertex. This algorithm is guaranteed to find a solution whose value is at most twice the optimal value.

Lee and Ho [13] also proposed a linear time algorithm with approximation ratio 2 for the weighted version of the problem, but the algorithm is based on the longest adjacent path removing heuristic.

Gu and Wang [14] proposed a 1.8 approximation algorithm for the undirected MCHEC problem based on the clockwise embedding scheme that was proposed in Ref. [12]. This algorithm has time complexity $O(m \cdot n)$. But this might not be optimal time if some hyperedges have $O(n)$ vertices and others just $O(c)$ vertices. For large values of k they employed a re-embedding of k hyperedges to reduce the maximum congestion L in the clockwise embedding $L - k$ to k, and for small values of k established a lower bound for the maximum congestion of an optimal embedding L^*. The approximation ratio is $(L - k)/L^*$, thus at most $(L - k)/\lceil L/2 \rceil$ because of $\lceil L/2 \rceil \le L^*$ from Ref. [12].

Lee and Ho [15] improve these results and presented an approximation algorithm with approximation ratio $1.5(opt + 1)$, where opt is the optimal value of maximum congestion for the undirected MCHEC problem by formulating the problem as an ILP and then by performing their LP-rounding algorithm (called clockwise (2/3)-rounding algorithm), which is based on two constraints: connectivity and capacity constraints.

Deng and Li [16] presented a polynomial-time approximation scheme (PTAS) for the undirected MCHEC problem. This algorithm is based on three algorithms for special cases of the problem. The first algorithm uses a combinatorial approach for the case when $m \leq C \cdot \log n$, for some $C > 0$, and a solution with $1 + \epsilon$ factor of the optimum is constructed in $O(n^{(C+1)/\epsilon})$ time for any given $\epsilon > 0$. The second algorithm is for the case when the optimal congestion is large (i.e., optimal congestion $\geq c \cdot m$ where $c > 0$ is a constant and $m \geq C \cdot \log n$). This algorithm uses LP-relaxation to apply the randomized rounding strategy. The last algorithm employs a hybrid version of the first two methods by applying the LP relaxation method proposed by Li et al. [17] for a string problem, to a specified subset of variables, instead of only a randomized rounding procedure, which will generate errors when the optimal congestion is small compared to m.

Li and Wang [18] proposed a polynomial-time approximation algorithm for the directed MCHEC problem by extending the method proposed in Ref. [16] and developed a technique to reduce the time complexity by a factor of $O(m)$. They proposed an algorithm for the case when there are $O(\log n)$ hyperedges based on choosing only $2r$ out of all the directed edges for each hyperedge h_j and cutting only a segment of vertices that contains one of the $2r$ selected edges. The time complexity of this algorithm is $O((2r)^{O(\log n)})$. They used this algorithm to construct a polynomial-time approximation scheme with ratio $1 + 1/r$, where r is a constant, that takes $O(n^{2r-1} \cdot n^{O(\log 2r)})$ time when $m = O(\log n)$.

Randomization is a powerful technique in finding approximate solutions to difficult problems in combinatorial optimization by solving a relaxation (usually LP relaxations or semidefinite programming relaxations) of a problem and then using randomization to return from the relaxation to the original optimization problem.

Derandomization can be applied by using standard techniques to yield deterministic polynomial-time algorithms that yield approximations as good as those given by the randomized algorithms they are derived from, even though the process of derandomization typically takes a relatively simple and clean randomized rounding procedure and turns it into a complex and generally slower deterministic algorithm.

Randomized rounding is a probabilistic method to convert a solution of a relaxed problem into an approximate solution to the original problem. Relaxation is an optimization problem with an enlarged feasible region and extended objective function compared with an original optimization problem.

Li and Wang also proposed another algorithm for the case when $c \geq O(\log m)$ and $c = O(m)$ using LP and randomized rounding approach where $m = O(C_{opt})$ (C_{opt} is the minimum congestion cost of an optimal embedding) and using the standard derandomization method to find a polynomial-time approximation for the case when $m \geq O(\log n)$ and $C_{opt} \geq O(m)$ where C_{opt} is large compared to m. They further proposed a general algorithm for the case when C_{opt} is small compared to m by decomposing the set of all hyperedges into two groups so as to find approximate embedding using different methods for the two groups to construct a polynomial-time approximation scheme for the problem.

In Section 70.2, we discuss the undirected MCHEC Problem with the objective of minimizing the maximum sharing of a link. We outline and analyze recent approximation algorithm work for the undirected MCHEC problem and related lower bounds for these problems. In Section 70.3, we discuss the directed version of MCHEC problem.

70.2 The Undirected MCHEC Problem

In this section, we discuss the undirected MCHEC problem. Given a hypergraph and a cycle on the node set, the undirected MCHEC problem is to find an embedding of the hypergraph such that the maximum congestion of any edge in the cycle is minimized. An embedding of the hypergraph specifies a connecting path for each hyperedge of the hypergraph.

Given a cycle $C = (V, E_c)$ consisting of n vertices labelled clockwise by 1 through n where $V = \{1, 2, \ldots, n\}$ is a set of vertices and $E_c = \{(i, i+1) | 1 \leq i \leq n-1\} \bigcup \{(n, 1)\}$ is a set of n undirected links or $E_c = \{e_i^+ = (i, i+1), e_i^- = (i+1, i) | i = 1, 2, 3, \ldots, n\}$ ($n+1$ denoted as 1) is a set of $2n$ directed edges, respectively. Each edge in the cycle refers to an undirected (in this section) or directed (in the next section) link.

A hypergraph $H = (V, E_h)$ with m hyperedges defined over the same set of vertices V where $E_h = \{h_1, h_2, \ldots, h_m\}$ is a set of m hyperedges and the hyperedge h_i consists of $|h_i|$ vertices for interconnecting these vertices. A connecting path $(c - path)$ P_i $(1 \le i \le m)$ for each hyperedge h_i denotes a path such that all nodes in h_i are in P_i. An embedding of the hypergraph specifies a c-path for each hyperedge of the hypergraph. The congestion of each link is the number of c-paths that contain the link. The undirected MCHEC (or directed MCHEC) problem is to find an embedding of the hypergraph such that the maximum congestion of any link in the cycle or the ring is minimized.

70.2.1 The Re-Embedding Algorithm

Gu and Wang [14] proposed a re-embedding (i.e., re-embed some hyperedges to reduce the maximum congestion) algorithm to find a solution bound to approximation ratio 1.8.

Let $l(i)$ denote the congestion of link i, L the maximum congestion in the clockwise embedding, L^* the maximum congestion in the optimal embedding, g_k the smallest link where $l(g_k) \ge L - 2k + 1$, and h_k the largest link where $l(h_k) \ge L - 2k + 1$ for integer k when $1 \le k \le \lfloor L/2 \rfloor$ and $0 \le g_k \le h_k < n - 1$.

Each re-embedding candidate w.r.t. k (i.e., with respect to k) is a hyperedge that has a node in segment $\langle 0, g_k \rangle$, a node in segment $\langle h_k + 1, n - 1 \rangle$, and no node in segment $\langle g_k + 1, h_k \rangle$. For this re-embedding candidate w.r.t. k, we embed it in such a way that the c-path does not contain any link i where $g_k \le i \le h_k$ and get $g_k + 1 \le g_k$, $h_k + 1 \ge h_k$, and $x_k + 1 \le x_k$ where x_k denotes the number of candidates w.r.t. k. This algorithm re-embeds the candidates w.r.t. k or $k + 1$ after finishing the clockwise embedding. The value of k is decreased by one starting from $k = \lfloor L/2 \rfloor$ if $x_k \ge k$ such that the c-path for each candidate does not contain any link i where $g_k \le i \le h_k$. Otherwise, k is decreased by one and the re-embedding process is repeated.

The re-embedding algorithm. Below we present the re-embedding algorithm with an approximate ratio 1.8 (even in the special case, i.e., $((L = 2$ or $L = 4)$ and $k = 0)$ or $(L = 12$ and $k = 1))$ was proposed by Gu and Wang in Ref. [14].

```
Step 1. Perform the clockwise embedding on the hypergraph.
Step 2. Find links $g_k$ and $h_k$, $k = 1, 2, \ldots, \lfloor L/2 \rfloor$, $x_k = 0$ if $k = \lfloor L/2 \rfloor + 1$ or $k = 0$.
Step 3. For $k := \lfloor L/2 \rfloor$ Step $-1$ until 1 do
            if $x \ge k$ then go to Step 4.
Step 4. If $x_k \ge k + 1$ and $x_{k+1} \ge 1$ then
            re-embed $k + 1$ candidates w.r.t. $k$ including at least one candidate w.r.t. $k + 1$
            else if $((L = 2$ or $L = 4)$ and $k = 0)$ or $(L = 12$ and $k = 1)$ then
                call Subprocedure Special Cases else
                re-embed $k$ candidates w.r.t. $k$.
            Return
```

Subprocedure special cases. Let s denote the link with L, E the set of hyperedges whose c-paths contain s in the clockwise embedding, and p_i the c-path for $e_i \in E$ that does not contain link s, the subprocedure special cases is as follows:

```
If $(L = 2$ or $L = 4)$ and $k = 0$ then
    for $\forall e_i \in E$ do
      if re-embedding $e_i$ by $p_i$ reduces the $L$ by 1 then
      re-embed $e_i$ by $p_i$ and return
      else for $\forall e_i, e_j \in E$ do
          if re-embedding $e_i$ by $p_i$ and $e_j$ by $p_j$ reduces the $L$ by 2 then
            re-embed $e_i$ by $p_i$ and $e_j$ by $p_j$ and return
          re-embed one candidate w.r.t. $k = 1$
Return
```

Let L_k denote the maximum congestion in the clockwise embedding; we have the following lemma:

Lemma 70.1 **(Gu and Wang [14])**

$L_k = L - k$ or $L_k = L - k - 1$.

Given $L_k = L - k$ or $L_k = L - k - 1$, define

W: the set of the hyperedges such that each hyperedge has a node in segment $\langle 0, g\rangle$, a node in segment $\langle g+1, h\rangle$, and a node in segment $\langle h+1, n-1\rangle$.

X: the set of the hyperedges such that each hyperedge has a node in segment $\langle 0, g\rangle$, has no node in segment $\langle g+1, h\rangle$, and has a node in segment $\langle h+1, n-1\rangle$.

Y: the set of the hyperedges such that each hyperedge has a node in segment $\langle 0, g\rangle$, a node in segment $\langle g+1, h\rangle$, and has no node in segment $\langle h+1, n-1\rangle$.

Z: the set of the hyperedges such that each hyperedge has no node in segment $\langle 0, g\rangle$, has a node in segment $\langle g+1, h\rangle$, and a node in segment $\langle h+1, n-1\rangle$.

A lower bound is derived:

Lemma 70.2 **(Gu and Wang [14])**

For any links g and h with $0 \le g < h < n-1$, $L^* \ge (2/3)|W| + (1/3)(|X| + |Y| + |Z|)$.

Using the lower bound given in the above lemma, a approximation ratio is derived for small k:

Theorem 70.1

The approximation ratio of Algorithm Re-embedding is bounded by 1.8.

70.2.2 The Clockwise (2/3)-Rounding Algorithm

A LP for any Integer Programming (IP) can be generated by taking the same objective function and same constraints but with the requirement that variables are integer replaced by appropriate continuous constraints. The LP Relaxation of the IP is the LP obtained by omitting all integer and 0–1 constraints on variables.

Lee and Ho [15] proposed a ρ-rounding and a $1.5(opt+1)$ approximation algorithm, where opt denotes the optimal maximum congestion for the MCHEC problem, by employing a clockwise (2/3)-rounding after an ILP formulation of the problem.

Let h_i denote an ordered sequence $(v_1^i, v_2^i, \ldots, v_{|h_i|}^i)$ with $v_1^i \le v_2^i \le, \ldots, \le v_{|h_i|}^i$ and $p(i, j)$ denote the jth clockwise adjacent path of hyperedge h_i between v_j^i and v_{j+1}^i.

Also let a set of binary variables $Y = y_{p(i,j)} | \forall i \in \{1, 2, \ldots, m\}\ j \in \{1, 2, \ldots, |h_i|\}$ denote an assignment of adjacent paths to embed the hypergraph H in the cycle C with $y_{p(i,j)} = 1$ if $p(i, j)$ is embedded, or with $y_{p(i,j)} = 0$ otherwise. The set $P(e)$ is used to denote the adjacent paths that pass through the link $e \in E_c$, and $\varphi = \max_{e \in E_c}\{\sum_{p(i,j) \in P(e)} y_{p(i,j)}\}$ denotes the maximum link congestion of an assignment Y. The problem is formulated as the following ILP, based on two constraints: connectivity and capacity constraints.

$\Phi(Y)$: minimize φ

Subject to

$$\sum_{1 \le j \le |h_i|} y_{p(i,j)} \ge |h_i| - 1, \quad \forall h_i \in E_h, i \in \{1, 2, \ldots, m\}$$

$$\sum_{p(i,j) \in P(e)} y_{p(i,j)} \le \varphi, \quad \forall e \in E_c$$

$$y_{p(i,j)} \in \{0, 1\}, \quad \forall i \in \{1, 2, \ldots, m\}, \quad j \in \{1, 2, \ldots, |h_i|\}$$

The ρ-rounding algorithm begins by solving optimally the LP relaxation of the above ILP formulation and finds the optimal solution $\Phi(Y^L)$ where $(Y^L) = [y_{p(i,j)}^L]$ and $0 \le y_{p(i,j)}^L \le 1$ with $y_{p(i,j)}^R = 1$ if

$y^L_{p(i,\,j)} \geq \rho$, or $y^R_{p(i,\,j)} = 0$ otherwise, and then output the approximate solution $\varphi^R = \Phi(Y^R)$ of the maximum congestion, where $Y^R = [y^R_{p(i,\,j)}]$. They proved the following two lemmas.

Lemma 70.3 **(Lee and Ho [15])**

Given $\min_{1\leq j\leq |h_i|}\{y^L_{p(i,\,j)}\} \leq 1-\rho, 0<\rho\leq 1$ *for* $\forall h_i \in E_h, i \in \{1, 2, \ldots, m\}$, *the maximum congestion of* ρ*-rounding algorithm is at most* $1/\rho$ *times the optimum.*

Lemma 70.4 **(Lee and Ho [15])**

For every hyperedge $h_i \in E_h, |h_i| \geq k, i \in \{1, 2, \ldots, m\}$, *the value of the k-th smallest variable in* $\{y^L_{p(i,\,j)} \mid 1 \leq j \leq |h_i|\}$ *is at least* $(k-1)/k$.

Lemma 70.4 is used to prove the following theorem.

Theorem 70.2 **(Lee and Ho [15])**

The maximum congestion of $(1/2)$*-rounding algorithm is at most twice the optimum.*

The problem of unsplittable load is defined as follows:

Given two fractions $y^M_{g(i,2)} = \frac{3}{2}y^L_{g(i,2)}$ and $y^M_{g(i,1)} = 1 - y^M_{g(i,2)}$, for each disconnected hyperedge $h_i \in D$, find an optimal rounding assignment of $y^R_{g(i,1)}$ and $y^R_{g(i,2)}$ such that the maximum load increment, denoted as Δ, over any physical link in the ring is minimized.

Let $D = \{h_i \mid \sum_{1\leq j\leq|h_i|} y^R_{p(i,\,j)} = |h_i| - 2, 1 \leq i \leq m\}$ denote the set of disconnected hyperedges in the output of the clockwise (2/3)-rounding algorithm, the problem of rounding one of the smallest two variables on $h_i \in D$ is translated into an unsplittable subproblem of the ring loading problem, which is studied in Refs. [8,19–22]. Lee and Ho extended the merging and sequential rounding techniques proposed in Refs. [8,20] from the ring loading problem for ensuring $\sum_{p(i,\,j)\in P(e)} y^R_{p(i,\,j)} \leq \frac{3}{2}(1 + \sum_{p(i,\,j)\in P(e)} y^L_{p(i,\,j)}) \leq \frac{3}{2}(opt + 1), \forall e \in E_c$ (opt is the optimal value of the problem). They proved that:

Lemma 70.5 **(Lee and Ho [15])**

Given $y^M_{g(i,2)} = \frac{3}{2}y^L_{g(i,2)}$ *and* $y^M_{g(i,1)} = 1 - y^M_{g(i,2)}$, *for each disconnected hyperedge* $h_i \in D$, *we have* $y^M_{g(i,1)} \leq \frac{3}{2}y^L_{g(i,1)} - y^R_{g(i,1)}$, $y^M_{g(i,2)} \leq \frac{3}{2}y^L_{g(i,2)} - y^R_{g(i,2)}$ *and* $y^M_{g(i,1)} \leq \frac{3}{2}y^L_{g(i,\,j)} - y^R_{g(i,\,j)}$, $3 \leq j \leq |h_i|$.

The clockwise (2/3)-rounding algorithm. The clockwise (2/3)-rounding algorithm proposed is as follows:

Step 1. Solve optimally the LP relaxation of the ILP formulation. Given $\Phi(Y^L)$ denote the optimal solution ($Y^L = [y^L_{p(i,\,j)}]$ and $0 \leq y^L_{p(i,\,j)} \leq 1$).

Step 2. For all i, j, let $y^R_{p(i,\,j)} = 1$ if $y^L_{p(i,\,j)} \geq 2/3$ or $y^R_{p(i,\,j)} = 0$, otherwise.

Step 3. Given $g(i, j)$ denote the adjacent path with the jth smallest LP variable for $h_i \in D$. If the adjacent path $g(i, 2)$ for the second smallest variable of h_i is connected around the ring from vertex s_i to vertex t_i in the clockwise direction, then assume that the adjacent path $g(i, 1)$ is connected in the other way around the ring from t_i to s_i. Moreover, we define $y^M_{g(i,2)} = \frac{3}{2}y^L_{g(i,2)}$ and $y^M_{g(i,1)} = 1 - y^M_{g(i,2)}$, respectively.

Step 4. Given the adjacent paths $g(i, j)$ and variables $y^M_{g(i,\,j)}$ for all $h_i \in D$, $j \in \{1, 2\}$, perform the merging procedure on D repeatedly until only crossing adjacent paths remain.

Step 5. Let S be the set of remaining hyperedges. For each $h_i \in S$, let $y^M_{s(i,1)}$ and $y^M_{s(i,2)}$, respectively, be the first and second variables of h_i in the clockwise order and let $r_1, r_2, \ldots, r_{|S|}$ denote the labels of hyperedges in S in the clockwise sequential order.

Step 6. Given $y^R_{s(r_1,1)} = 1$ and $y^R_{s(r_1,2)} = 0$ if $y^M_{s(r_1,1)} > 0.5$, otherwise, $y^R_{s(r_1,1)} = 0$ and $y^R_{s(r_1,2)} = 1$.

Step 7. For $\forall k$, $2 \leq k \leq |S|$, if $(1 - y^M_{s(r_k,1)}) + \sum_{i\in\{r_1,\,\ldots,\,r_{k-1}\}}(y^R_{s(i,1)} - y^M_{s(i,1)}) < \frac{1}{2}$ then set $y^R_{s(r_k,1)} = 1$ and $y^R_{s(r_k,2)} = 0$ else set $y^R_{s(r_k,1)} = 0$ and $y^R_{s(r_k,2)} = 1$. Due to the complementary nature, we have $\sum_{i\in\{r_1,\ldots,r_k\}}(y^R_{s(i,1)} - y^M_{s(i,1)}) \in [-\frac{1}{2}, \frac{1}{2})$ and $\sum_{i\in\{r_1,\ldots,r_k\}}(y^R_{s(i,2)} - y^M_{s(i,2)}) \in (-\frac{1}{2}, \frac{1}{2}]$.

Step 8. Output the approximate solution $\varphi^R = \Phi(Y^R)$ of the maximum congestion.

Finally, they proved the following theorem that the clockwise (2/3)-rounding algorithm has an approximation bound of $1.5(opt + 1)$, where *opt* represents the optimal value of the problem.

Theorem 70.3 (Lee and Ho [15])

The clockwise (2/3)-rounding algorithm has an approximation bound of $1.5(opt + 1)$, *where opt represents the optimal value of the problem.*

70.2.3 The Randomized Rounding Algorithm

Randomization is a powerful technique in finding approximate solutions to difficult problems in combinatorial optimization by solving a relaxation (usually LP relaxations or semidefinite programming relaxations) of a problem and then using randomization to return from the relaxation to the original optimization problem.

Deng and Li [16] proposed a polynomial-time approximation scheme to solve the problem for a special case with $O(\log\ n)$ hyperedges and for a case with large optimal solution. For the special case with $O(\log\ n)$ hyperedges, given $m \leq C(\log\ n)$ for any fixed constant $C > 0$ and $x_j = E_C - E_{l_j}^{(j)}, 1 \leq l_j \leq k_j$, denoting an embedding of hyperedge b_j for $j = 1, 2, \ldots, m$, the proposed algorithm is to enumerate all the possible solutions for each r-element subset $\{e_{i_1}, e_{i_2}, \ldots, e_{i_r}\}$ of the n input edges in E_C and output the best solution.

Let $E_l^{(j)} = \{e_{i_l}^{(j)}, e_{i_l+1}^{(j)}, \ldots, b_{i_{l+1}-1}^{(j)}\}$ denote an embedding of the hyperedge $e_j = \{i_1^{(j)}, i_2^{(j)}, \ldots, i_{k_j}^{(j)}\}$ which partitions the edges on the cycle C into k_j segments represented by $E_l^{(j)}, l = 1, 2, \ldots, j$. An x-embedding $x = (x_1, x_2, \ldots, x_m)$ denotes a vector of dimension m, where x_j is a subset of edges in C that forms an embedding of jth hyperedge b_j (i.e., $x_j = E_C - E_{l_j}^{(j)}$).

Let some l_j, $1 \leq l_j \leq k_j$, denote that the c-path embeds b_j excluding $E_{l_j}^{(j)}$, $1 \leq j \leq m$, e_i denote an edge of the cycle C, and $e_i(x)$ denote the congestion of edge e_i for the feasible solution x-embedding, the MCHEC problem is formulated as the following ILP:

$$
\begin{aligned}
&\text{Minimize} && z \\
&\text{Subject to} && e_i(x) \leq z, \quad i = 1, 2, \ldots, n
\end{aligned}
$$

Also let $A(I)$ denote the cost of the solution found by A, $OPT(I)$ denote the cost of an optimal solution with a polynomial-time complexity on the input size if ϵ is considered as a constant, the objective is to find an approximation algorithm A in a polynomial time that is bound to the following performance ratio because it was known to be NP-complete:

$$R_A(I, \epsilon) = \frac{A(I)}{OPT(I)} \leq 1 + \epsilon$$

For small number of hyperedges, they proved the following theorem:

Theorem 70.4 (Deng and Li [16])

The MCHEC problem can be solved by a PTAS when $m < C \cdot \log\ n$ *for any constant* $C > 0$. *In particular, for any given* $\epsilon > 0$, *a solution with* $1 + \epsilon$ *factor of the optimum can be found in time* $O(n^{(C+1)/\epsilon})$.

Let us consider the case when the optimal solution is large. Let C_{opt} denote the (large) objective value of optimum solution to the following ILP, where $c_{opt} \geq c \cdot m$ with a constant $c > 0$. The binary variable $x_{j,l}$ has the value 1 if $x_j = E_C - E_l^{(j)}$, and value 0 otherwise, where $1 \leq j \leq m$ and $1 \leq l \leq k_j$. The set $\widehat{x}_j(e_i, l)$ denotes the index functions with value 0 if $e_i \in E_l^{(j)}$, or value 1 otherwise. The undirected MCHEC problem is formulated as the following ILP:

$$
\begin{aligned}
&\text{Minimize} && z \\
&\text{Subject to} && \sum_{l=1}^{k_j} x_{j,l} = 1, \quad j = 1, 2, 3, \ldots, m \\
&\sum_{j=1}^{m} \sum_{l=1}^{k_j} \widehat{x}(e_i, l) \cdot x_{j,l} \leq z, && i = 1, 2, 3, \ldots, n
\end{aligned}
$$

For large optimal solution, they proved the following theorem:

Theorem 70.5 (Deng and Li [16])

The MCHEC problem can be solved with a PTAS when $C_{opt} \geq c \cdot m$, and m is sufficiently large (by choosing sufficiently large constant C such $m \geq C \cdot \log\ n$).

Deng and Li [16] also employed the randomized rounding strategy to round a fractional optimal solution $\overline{x}_{j,l}$; $j = 1, 2, \ldots, m$; $l = 1, 2, \ldots, k_j$, for each $j = 1, 2, \ldots, m$, independently, with probability $\overline{x}_{j,l}$, to set $\widehat{x}_{j,l} = 1$ and $\widehat{x}_{j,h} = 0$ for any $h \in 1, 2, \ldots, k_j - l$ so as to find a solution for the problem.

Let $R_{i_1,i_2,\ldots,i_k}$ denote the set of indices of hyperedges such that $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ are all relative with respect to those hyperedges (i.e., $R_{i_1,i_2,\ldots,i_k} = \{1 \leq j \leq m | \exists\, l_j \in 1, 2, \ldots, k_j$ such that $e_{i_1}, e_{i_2}, \ldots, e_{i_k} \in E_{l_j}^{(j)}\})$, $y|\,R_{i_1,i_2,\ldots,i_k}$ denote a partial embedding of y restricted on $R_{i_1,i_2,\ldots,i_k}$, where $U_{i_1,i_2,\ldots,i_k} = \{1, 2, \ldots, m\} - R_{i_1,i_2,\ldots,i_k}$, the problems is formulated as the following ILP:

$$\begin{aligned} &\text{Minimize} \quad z \\ &\text{Subject to} \quad \sum_{l=1}^{k_j} x_{j,l} = 1, \qquad j = 1, 2, 3, \ldots, |U| \\ &\sum_{j=1}^{|U|} \sum_{l=1}^{k_j} \widehat{x}(e_i, l) \cdot x_{j,l} \leq z - e_i(x^{(i_1)}|_R), \qquad i = 1, 2, 3, \ldots, n \end{aligned}$$

The Randomized Rounding Algorithm. The proposed randomized rounding algorithm has a fractional solution $\overline{x}_{j,l}$, $1 \leq j \leq |U|$ with cost $\overline{d} \leq (1 + \frac{1}{r-1}) \cdot C_{opt}$ and is as follows:

Step 1. For each r-element subset $\{e_{i_1}, e_{i_2}, \ldots, e_{i_k}\}$ of the n input edges in E_C do

(a) $R = \{1 \leq j \leq m | e_{i_1}, e_{i_2}, \ldots, e_{i_k}\}$ are in the same segment of jth hyperedge, $U = \{1, 2, \ldots, m\} - R$.

(b) For the hyperedges with indices in $R_{i_1,i_2,\ldots,i_r}$, take $x^{(i_1)}|_R$ as an approximation of optimal embedding x.

(c) For the hyperedges with their indices in $U_{i_1,i_2,\ldots,i_r}$, find a partial embedding $\widehat{x}|_U$

(d) Get an approximation $\hat{x}$ of x by concatenating $x^{(i_1)}|_R$ and $\widehat{x}|_U$.

Step 2. Output the best solution obtained in Step 1.

For the general MCHEC problem, they proved the following theorem:

Theorem 70.6 (Deng and Li [16])

There is a PTAS for the MCHEC problem.

70.3 The Directed MCHEC Problem

Let $c(e_i^+)$ and $c(e_i^-)$ denote the congestion on the directed ring, $h = (u, S)$ a directed hyperedge, u the source of hyperedge, $S \subseteq V - u$ the set of sinks, and $h_j = (u_j, S_j)$ a hyperedge in E_h.

Also let $S_j = \{i_1^j, i_2^j, i_3^j, \ldots, i_j^k\}$ denote the set of jth sink (i_0^j regarded as u_j), $k_j = |S_j|$ the total number of sink vertices in the hyperedge, P the segment of vertices from vertex i_k^j to vertex $i_{k+1}^j (k = 0, 1, 2, \ldots, k_j - 1)$, and $P_{k_j}^j$ the segment of vertices from vertex i_k^j to vertex i_0^j ($k = 0, 1, 2, \ldots, k_j - 1$). An embedding of h_j on the ring (i.e., a P_k^j embedding, if it is cut, then there are $k_j + 1$ different embedding) is formed by cutting one of the paths P_k into two directed paths both starting from k, where $k = 0, 1, 2, \ldots, k_j$.

Given an embedding x of all the hyperedges E_H, the congestion $e_i^+(x)$ or $e_i^-(x)$ of a directed edge is the number of times that the edge e_i^+ or e_i^- is used in the embedding. The directed MCHEC problem is to find an embedding for each $b_j \in E_h$ such that every edge e_i^+ and e_i^- is used at most c times and c is minimized.

Li and Wang [18] proposed a polynomial-time approximation scheme for the problem of embedding directed hyperedges on a directed ring. They deal with a special case where the number of hyperedges $m = O(\log\ n)$ and solve the case where the number of hyperedges $m = O(C_{opt})$ with C_{opt} denoting the minimum congestion cost of an optimal embedding. They also proposed an algorithm with approximation ratio $1 + \frac{1}{r}$ (r is a constant) for the case of $O(\log\ n)$ hyperedges by cutting a P_k^j, which contains one of the $2r$ edges and by choosing one of the $2r$ edges for each hyperedge h_i.

Let $(i_1, i_1 + 1), (i_2, i_2 + 1), (i_3, i_3 + 1), \ldots, (i_{2r}, i_{2r} + 1)$ denote the $2r$ edges represented by indices $i_1, i_2, i_3, \ldots, i_{2r}$, $x = (x_1, x_2, x_3, \ldots, x_m)$ denote an embedding of H where x_j represents the selection of P_k^j cut for the embedding of b_j, and $E_j(x)$ denote the segment P_k^j cut for the embedding x of b_j.

Also let $Q_{i_1,i_2,\ldots,i_{2r}}(x)$ denote the set of indices of hyperedges such that j is in $Q_{i_1,i_2,\ldots,i_{2r}}(x)$ if and only if $E_j(x)$ contains at least one of the $2r$ edges and H_r denote the set of the (at most $2c$) h_j's with $e_1^+ \in E_j(x)$. Li and Wang [18] proposed the following embedding algorithm. The algorithm has $O((2r)^{O(\log\ n)})$ time complexity.

The Embedding Algorithm

Step 1. For each remaining edge e_g^+ do

Step 2. If there are more than $\frac{c}{r}$ hyperedges $b_j \in H_r$ with $e_g^+ \in E_j(x)$, then select the index q and set $H_r - \{j | e_g^- \in E_j(x)\}$

Step 3. If the size of H_r is more than $\frac{c}{r}$ than goto Step 1 else stop.

Step 4. Try all possible choices of $i_2, i_3, \ldots, i_{2r}$ with time complexity $O(n^{2r-1})$.

Step 5. For each $b_j \in H$, try the $2r - 1$ choices for cutting the path (cut a path and obtain two directed paths to form an embedding of a hyperedges on the ring) P_k^j containing $e_{i_1}^+, e_{i_2}^+, \ldots, e_{i_2r}^+$ with time complexity $O((2r)^m) = O((2r)^{O(\log\ n)}) = n^{O(\log 2r)}$.

By using an enumerating method for $m = O(\log\ n)$ they proved the following theorem.

Theorem 70.7 **(Li and Wang [18])**

There is a PTAS with ratio $1 + (1/r)$ *that runs in* $O(n^{2r-1} \cdot n^{O(\log\ 2r)})$ *when* $m = O(\log\ n)$.

For the case when $c = O(m)$ and $c \geq O(\log\ n)$ their approach is as follows. Let $h_j = (u_j, S_j)$ denote a hyperedge, $x_{j,1}, x_{j,2}, \ldots, x_{j,k_j+1}$ where the binary variables with $x_{j,l} = 1$ if P_l^j is a cut for the embedding of b_j, or value 0 otherwise. Let $\mu_{i,q,j}$ denote the binary variables for each segment P_q^j of b_j and an edge e_i^+ with value 1 if the edge e_i^+ is in the segment P_q^j of b_j, or value 0 otherwise. The problem is formulated as the following ILP and is solved in a polynomial-deterministic time by using the randomized rounding approach and standard derandomization method for packing integer programs proposed in Ref. [23].

$$\text{Minimize } c$$

$$\text{Subject to } \sum_{l=1}^{k_j+1} x_{j,l} = 1$$

$$c(e_i^+) = \sum_{j=1}^{m} \sum_{q=1}^{k_j+1} \mu_{i,q,j}(x_{j,q+1} + x_{j,q+2} + \cdots + x_{j,k_j}) \leq c$$

$$c(e_i^-) = \sum_{j=1}^{m} \sum_{q=1}^{k_j+1} \mu_{i,q,j}(x_{j,0} + x_{j,1} + \cdots + x_{j,q-1}) \leq c$$

Let C_{opt} denote the optimal congestion of the above ILP formulation, similar to Lemma 3 in Ref. [16], they proved the following theorem:

Theorem 70.8 **(Li and Wang [18])**

Assume that $m \geq c_1 \cdot \log\ n$ *and* $C_{opt} = c_2 \cdot m$. *Let* $\hat{x}$ *be the* $0 - 1$ *solution obtained by randomized rounding. With probability at least* $1 - n^{1-(1/3)\epsilon^2 \cdot c_2^2 \cdot c_1}$, *for any* e_i^+ *and* e_i^- *in* E_h, $e_i^+(\hat{x}) \leq (1+\epsilon)C_{opt}$, *and* $e_i^-(\hat{x}) \leq (1+\epsilon)C_{opt}$.

Using the standard de-randomization method for packing integer program [23], they proved the following theorem:

Theorem 70.9 (Li and Wang [18])

There is a PTAS for the directed MCHEC problem when $m \geq O(\log n)$ *and* $C_{opt} \geq O(m)$.

They proposed a general algorithm for the case when C_{opt} is small compared to m. The set of all hyperedges is decomposed into two groups. Each group is embedded using a different method.

Let $e^+_{i_1}, e^+_{i_2}, \ldots, e^+_{i_2r}$ denote the $2r$ edges on the ring, $R_{i_1,i_2,\ldots,i_{2r}} = \{1 \leq j \leq m | \exists l, e^+_{i_k} \in P^j_l, \forall k \in \{1, 2, \ldots, 2r\}\}$, $U_{i_1,i_2,\ldots,i_{2r}} = \{1, 2, \ldots, m\} - R_{i_1,i_2,\ldots,i_{2r}}$, x_{opt} denote an optimal embedding, $x_{opt}|R_{i_1,i_2,\ldots,i_r}$ and $x_{opt}|U_{i_1,i_2,\ldots,i_r}$ denote the reduced embedding of x_{opt} on the sets of $R_{i_1,i_2,\ldots,i_r}$ and $U_{i_1,i_2,\ldots,i_r}$, respectively, and $c(e^+_i|R)$ and $c(e^-_i|R)$ denote the embedding of b'_js in $R_{i_1,i_2,\ldots,i_{2r}}$, the problem is formulated as the following ILP:

Minimize c

Subject to

$$\sum_{l=1}^{k_j+1} x_{j,l} = 1, \quad j = 1, 2, 3, \ldots, |U_{i_1,i_2,\ldots,i_{2r}}|$$

$$\sum_{j=1}^{|U_{i_1,i_2,\ldots,i_{2r}}|} \sum_{q=1}^{k_j+1} \mu_{i,q,j}(x_{j,q+1} + x_{j,q+2} + \cdots + x_{j,k_j}) \leq c - c(e^+_i|R)$$

$$\sum_{j=1}^{|U_{i_1,i_2,\ldots,i_{2r}}|} \sum_{q=1}^{k_j+1} \mu_{i,q,j}(x_{j,0} + x_{j,1} + \cdots + x_{j,q-1}) \leq c - c(e^-_i|R)$$

This algorithm computes $U_{i_1,i_2,\ldots,i_r}$ and $R_{i_1,i_2,\ldots,i_{2r}}$ by employing the enumerating approach discussed above to find an embedding of the set of hyperedges in $U_{i_1,i_2,\ldots,i_r}$ and by cutting the ring at edge e^+_1 of the hyperedges in $R_{i_1,i_2,\ldots,i_r}$ for the case when $|U_{i_1,i_2,\ldots,i_{2r}}| \leq C \cdot \log n$. This algorithm employs the LP randomized rounding approach also discussed above to find an embedding of the set of hyperedges in $U_{i_1,i_2,\ldots,i_{2r}}$ by cutting the ring at edge e^+_1 of the hyperedges in $R_{i_1,i_2,\ldots,i_r}$ for the case when $|U_{i_1,i_2,\ldots,i_r}| > C \cdot \log n$.

70.4 Discussion

The multicast congestion problems are critical to many network applications in multimedia streamlining such as multimedia distribution, software distribution, and video-conference; groupware system; game communities; and electronic design automation such as routing nets around a rectangle and moat routing. All the approximation algorithms discussed above have been developed based on the deep analysis of the problems from different points of views, which thus have been formulated as a variety of ILPs and solved by employing different schemes.

The generalization of the problem of finding edge disjoint paths is a NP-hard combinatorial problems, i.e., minimum multicast Steiner tree problem, and also a max-SNP hard problem, i.e., there is a constant $\epsilon > 0$ such that it is NP-hard to find a $(1 + \epsilon)$ approximation. The main problem with the LP-relaxation is the time required to solve the ILP. To improve the efficiency, it is mandatory to make better use of the multicast congestion parameters in order to obtain a simplified approximation algorithm with tighter approximation bound. Finding better and fast approximation algorithms for specific classes of hypergraphs is worth further investigation as well.

References

[1] Vempala, S. and Vöcking, B., Approximating multicast congestion, *Proc. ISAAC*, 1999, p. 367.

[2] Raghavan, P. and Thompson, C., Multiterminal global routing: a deterministic approximation scheme, *Algorithmica*, 6, 73, 1991.

[3] Carr, R. and Vempala, S., Randomized metarounding, *Proc. STOC*, 2000, p. 58.

[4] Chen, S., G*ü*nl*ü*k, O., and Yener, B., The multicast packing problem, *IEEE/ACM Trans. Network.*, 8(3), 311, 2000.

[5] Fagin, R. and Stockmeyer, L., Relaxing the triangle inequality in pattern matching, *Int. J. Comput. Vision*, 28(3), 219, 1988.

[6] Frank, A., Edge-disjoint paths in planar graphs, *J. Comb. Theory Ser. B*, 38, 164, 1985.

[7] Okamura, H., and Seymour, P. D., Multicommodity flows in planar graph, *J. Comb. Theory Ser. B*, 31, 75, 1981.

[8] Schrijver, A., Seymour, P., and Winkler, P., The ring loading problem, *SIAM Disc. Math.*, 11(1), 1, 1998.

[9] Schrijver, A., Seymour, P., Winkler, P., The ring loading Problem., *SIAM Rev.*, 41(4), 777, 1999.

[10] Ganley, J. L. and Cohoon, J. P., Minimum-congestion hypergraph embedding on a cycle, *IEEE Trans. Comput.*, 46(5), 600, 1997.

[11] Gonzalez T., Improved approximation algorithms for embedding hyperedges in a cycle, *Inf. Proc. Lett.*, 67, 267, 1998.

[12] Carpenter, T., Cosares, S., Ganley, J. L., and Saniee, I., A simple approximation algorithm for two problems in circuit design, *IEEE Trans. Comput.*, 47(11), 1310, 1998.

[13] Lee, S. L. and Ho, H. J., Algorithms and complexity for weighted hypergraph embedding a cycle, *Proc. of the Int. Symp. on Cyber World*, 2002, p. 70.

[14] Gu, Q. P. and Wang, Y., Efficient algorithm for embedding hypergraph in a cycle, *Proc. Int. Conf. on High Perf. Comput.*, 2003, p. 85.

[15] Lee, S. and Ho, H. J., A 1.5 approximation algorithm for embedding hyper-edges in a cycle, *IEEE Trans. Parallel Dist. Syst.*, 16(6), 481, 2005.

[16] Deng, X. and Li, G., A PTAS for embedding hypergraph in a Cycle (extended abstract), *Proc. Int. Colloquium on Automata, Lang. and Prog.*, 2004, p. 433.

[17] Li, M., Ma, B., and Wang, L., On the closest string and substring problems, *JACM* 49(2), 157, 2002.

[18] Li, K. and Wang, L., A polynomial time approximation scheme for embedding a directed hypergraph on a ring, *Proc. AAIM*, 2005, p. 392.

[19] Cosares, S. and Saniee, I., An optimization problem related to balancing loads on SONET rings, *Telecommun. Syst.*, (3), 165, 1992.

[20] Wilfong, P. and Winkler, P., Ring routing and wavelength translation, *Proc. SODA*, 1998, p. 333.

[21] Khanna, S., A polynomial time approximation scheme for SONET ring loading problem, *Bell Labs Tech. J.*, 36, 36–41, 1997.

[22] Myung, Y. S., An efficient algorithm for the ring loading problem with integer demand splitting, *SIAM J. Disc. Math.*, 14(3), 291, 2001.

[23] Motwani, R., and Raghavan, P., *Randomized Algorithms*, Cambridge University Press, Cambridge, 1995.

71

QoS Multimedia Multicast Routing

Ion Măndoiu
University of Connecticut

Alex Olshevsky
Massachusetts Institute of Technology

Alexander Zelikovsky
Georgia State University

71.1 Introduction

Recent progress in audio, video, and data storage technologies has given rise to a host of high-bandwidth real-time applications such as videoconferencing. These applications require Quality of Service (QoS) guarantees from the underlying networks. Thus, multicast routing algorithms, which manage network resources efficiently and satisfy the QoS requirements, have come under increased scrutiny in recent years [1]. The focus on multimedia data transfer capability in networks is expected to further increase as videoconferencing applications gain popularity.

It is becoming apparent that new network mechanisms will be required to provide differentiated quality guarantees to network users. Of particular importance is the problem of optimal multimedia distribution from a source to a collection of users with heterogeneous demands. Multimedia distribution is usually done via multicast trees. There are two main reasons for using trees in multicast routing: (a) data can be transmitted concurrently to destinations along the branches of the tree, and (b) only a minimum number of copies of data must be transmitted since information replication is limited to the branching points of the tree [2]. The bandwidth savings obtained from the use of multicast trees can be maximized by using optimal or nearly optimal multicast tree algorithms, and future networks are expected to integrate such algorithms into their operation [3].

Several versions of the QoS multicast problem have been studied in the literature. These versions seek routing tree cost minimization subject to (1) end-to-end delay, (2) delay variation, and/or (3) minimum bandwidth constraints (see, e.g., Refs. [3–5]). This chapter deals with the case of minimum bandwidth constraints, that is, the problem of finding an optimal multicast tree when each terminal possesses a different rate of receiving information. This problem is a generalization of the classical Steiner tree problem and therefore NP-hard [6]. Formally, given a graph $G = (V, E)$, a source s, a set of terminals S, and two functions: *length* : $E \to R^+$ representing the length of each edge and *rate* : $S \to R^+$ representing the rate of each terminal, a *multicast tree* T is a tree in G spanning s and S. The *rate* of an edge e in a multicast tree T, denoted by *rate*(e, T), is the maximum rate of a downstream terminal, that is, of a terminal in the connected component of $T - e$, which does not contain s. The *cost* of a multicast tree T is defined as

$$cost(T) = \sum_{e \in T} length(e) \cdot rate(e)$$

Quality of Service Multicast Tree (QoSMT) Problem
Given a network $G = (V, E, length, rate)$ with source $s \in V$ and set of terminals $S \subseteq V$, find a minimum cost multicast tree in G.

Without loss of generality, in this chapter we further assume that the rates belong to a given discrete set of possible rates: $0 = r_0 < r_1 < \cdots < r_N$. The QoSMT problem is equivalent to the Grade of Service Steiner Tree problem [7], which has a slightly different formulation: in the latter the network has no source node and rates r_e must be assigned to edges so that the minimum edge rate on the tree path from a terminal with rate r_i to a terminal with rate r_j is at least $min(r_i, r_j)$, and such that the total tree cost is minimized. A more general *QoSMT with priorities* was considered by Charikar et al. [6]. In this version of the problem the cost of an edge e is given arbitrarily instead of being equal to the length times the rate. In other words, edge costs in QoSMT with priorities are not required to be proportional to edge rates. This generalization seems more difficult—the best known approximation ratio is logarithmic (which also holds for multiple multicast groups) [6].

The QoSMT problem was introduced in the context of multimedia distribution over communication networks by Maxemchuk [5]. Maxemchuk suggested a low-complexity heuristic that can be used to build reliable multicast trees in many practical applications. Following Maxemchuk, Charikar et al. [6] gave a useful approximation algorithm that finds a solution within $e\alpha$ of the optimal, where $\alpha < 1.550$ is the best approximation ratio of an algorithm for the Steiner tree problem. This is the first known algorithm with a constant approximation ratio for this problem. Recently, an approximation ratio of 3.802 based on accurate estimation of Steiner tree length has been achieved [8].

We note that the QoSMT problem was also considered previously (under different names) in the operations research literature. A number of results for particular instances of the problem were obtained: Current et al. [9] gave an integer programming formulation for the problem and proposed a heuristic algorithm for its solution. Results for the case of a series-parallel graph were presented in Ref. [25]. Some results for the case of few rates were obtained by Balakrishnan et al. [10,11]. Specifically, Ref. [11] (see also Ref. [7]) suggested an algorithm for the case of two nonzero rates with approximation ratio of $\frac{4}{3}\alpha < 2.066$. An improved approximation algorithm with a ratio of 1.960 was proposed in Ref. [8]. For the case of three nonzero rates, Mirchandani [12] gave an 1.522-approximation algorithm.

This chapter is organized as follows. First, we describe centralized algorithms for the QoSMT problem, spending the bulk of the time on the algorithms in Ref. [8], which have best approximation factors to date. Although these algorithms have superior quality, they cannot be easily adjusted for operation in a distributed environment. Thus, we then describe a more practical primal-dual approach to the QoSMT problem following Ref. [13]. This approach yields algorithms that have natural distributed implementations, and work well even when the multimedia source does not have exact knowledge of network topology. We conclude with an experimental comparison showing the advantage of the primal-dual approach over practical heuristics proposed in the literature.

71.2 Centralized Approximation Algorithms

Table 71.1 and Table 71.2 summarize the approximation ratios of known centralized algorithms for the QoSMT problem, for the cases of two nonzero rates and unbounded number of nonzero rates, respectively. In this table, we present the approximation ratios achievable using various Steiner tree approximation algorithms as a subroutine. Note that along with the best approximation ratios resulting from the use of the loss-contracting Steiner tree algorithm in Ref. [14], we also give approximation ratios resulting from the use of the more practical Steiner tree algorithms from Refs. [15–18]. In this section, we briefly discuss Maxemchuk's [5] and Charikar et al.'s [6] methods, and then give a detailed description of best-to-date β-convex approximation algorithms of Karpinski et al. [8].

TABLE 71.1 QoSMT Problem with Two Rates

Steiner Tree Algorithm	LCA [14]	LCA + RNS [15]	BR [16,18]	MST [17]
Runtime	Polynomial	Polynomial	$O(n^3)$ [19]	$O(n \log n + m)$ [20]
Approximation ratio	$\frac{4}{3}\frac{1+\ln 3}{2} + \epsilon$	$\frac{20}{9} + \epsilon$	$\frac{22}{9}$	$\frac{8}{3}$
in Refs. [7,11]	$< 2.066 + \epsilon$	$< 2.223 + \epsilon$	< 2.445	< 2.667
Improved ratio [8]	—	$1.960 + \epsilon$	2.237	2.414

Note: Runtime and approximation ratios of previously known algorithms and of the algorithms given in this chapter. In the runtime, n and m denote the number of nodes and edges in the original graph $G = (V, E)$, respectively. Approximation ratios associated with polynomial-time approximation schemes are accompanied by a $+\epsilon$ to indicate that they approach the quoted value from above and do not reach this value in polynomial time.

TABLE 71.2 Approximation Ratios for QoSMT Problem with an Arbitrary Number of Rates

Steiner Tree Algorithm	LCA [14]	RNS [15]	BR [16,18]	MST [17]
Approximation	$e\frac{1+\ln 3}{2} + \epsilon$	$e\frac{5}{3} + \epsilon$	$e\frac{11}{6}$	$2e$
ratio in [15]	$< 4.212 + \epsilon$	$< 4.531 + \epsilon$	< 4.984	< 5.44
Improved ratio [8]	—	$3.802 + \epsilon$	4.059	4.311

71.2.1 Maxemchuk's Algorithm

Maxemchuk [5] introduced the QoSMT problem and proposed the first heuristic to solve this problem. His algorithm is a modification of the Minimum Spanning Tree (MST) heuristic for Steiner trees [17] (see Figure 71.1).

The extensive experiments given in Ref. [5] demonstrate that this method works well in practice. Nevertheless, the following example shows that the method may produce arbitrarily large error (linear in the number of rates) compared with the optimal tree. Consider the natural generalization of the example in Figure 71.2 with an arbitrary number k of distinct rates. Its optimal solution has a cost of about 1, whereas Maxemchuk's method returns a solution of cost about $(k+1)/2$. As there are $2^{k-1}+1$ nodes, this cost can also be written as $1 + \frac{1}{2}\log_2(n-1)$, where n is the number of nodes in the graph. We conclude that the approximation ratio of Maxemchuk's algorithm is no better than linear in the number of rates and no better than logarithmic in the number of nodes in the graph.

71.2.2 The Charikar–Naor–Schieber Algorithms

Charikar et al. [6] gave the first constant-factor approximation algorithms for the QoS Steiner tree problem. The simplest version is a *binary rounding* algorithm. In its first step, all rates are rounded to the closest

Input: A graph $G = (V, E, length, rate)$ with a source s in V and a collection of terminals $S \subseteq V$.

Output: A QoSMT spanning the source and the terminals.

1. Initialize the current tree to $\{s\}$.
2. Find a nonreached terminal t of highest rate with the shortest distance to the current tree.
3. Add t to the current tree along with a shortest path connecting it to the current tree.
4. Repeat until all terminals are spanned.

FIGURE 71.1 Maxemchuk's algorithm for the QoSMT problem.

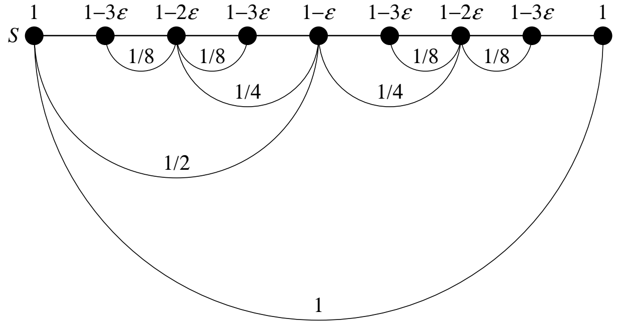


FIGURE 71.2 A bad example for Maxemchuk's algorithm, with $k = 4$ rates. In the figure, $\varepsilon = 1/2^{2k-1}$. The rate of each node is given above the node. The edge lengths are given on the thin curved arcs, while on the solid horizontal line each segment has length $1/2^{k-1} + \varepsilon$. The optimum, of total cost $1 + 2^{k-1}\varepsilon = 1 + 2^{k-1}(1/2^{2k-1}) = 1 + 1/2^k$, uses the solid horizontal line at rate 1. Maxemchuk's algorithm picks the thin curved arcs at a cost of $1 + (1/2)(1 - \varepsilon) + 2(1/4)(1 - 2\varepsilon) + 4(1/8)(1 - 3\varepsilon) \geq ((k + 1)/2)(1 - 1/2^k)$.

power of 2 to produce the rounded up instance of this problem (clearly, this at most doubles the cost of an optimal solution). In its second step, Steiner trees are computed separately for each rate (within some approximation ratio α). The union of these trees is the final solution.

Consider the network obtained by replacing each edge of rate 2^i in an optimal solution by $i + 1$ parallel edges of rates $2^0, 2^1, \ldots, 2^{i-1}, 2^i$, respectively. In the new network, edges of a specific rate form a Steiner tree spanning all terminals of the respective rate. Since the optimal cost in this new network is no more than twice the cost of the rounded up instance, taking the union of all the computed Steiner trees introduces another factor of 2 to the approximation ratio. Thus the final approximation factor is $2 \times (2\alpha) = 4\alpha$.

Using a randomization technique, Charikar, et al. [6] reduce the approximation ratio to $e\alpha \approx 4.21$, where $e \approx 2.71$ is the Euler constant and $\alpha \approx 1.55$ is the currently best approximation ratio for the Steiner tree problem.

71.2.3 β-Convex Steiner Tree Approximation Algorithms

In this section, we introduce the notion of β-convex Steiner tree approximation algorithms and show tighter upper bounds on their output when applied to the QoSMT problem.

We begin by reviewing some Steiner tree definitions. A Steiner tree is a minimum-length tree connecting a subset of the graph's nodes. The nodes in a subset are usually referred to as *terminal* nodes. A Steiner tree is called *full* if every terminal is a leaf. A Steiner tree can be decomposed into components, which are full by breaking the tree up at the nonleaf terminals. A Steiner tree is called *k-restricted* if every full component has at most k terminals. Let us denote the length of the optimum k-restricted Steiner tree as opt_k and the length of the optimum unrestricted Steiner tree as opt. Let the k-restricted Steiner ratio ρ_k be $\rho_k = sup\frac{opt_k}{opt}$, where the supremum is taken over all instances of the Steiner tree problem. It has been shown in Ref. [21] that $\rho_k = \frac{(r+1)2^r+s}{r2^r+s}$, where r and s are obtained from the decomposition $k = 2^r + s$, $0 \leq s < 2^r$. A slightly tighter bound on the length of the optimal k-restricted Steiner tree has been established in Ref. [8].

Theorem 71.1 (Karpinski et al. [8])

For every Steiner tree T partitioned into edge-disjoint full components T^i,

$$opt_k \leq \sum_i (\rho_k(l(T^i) - D(T^i)) + D(T^i))$$

where $l(T^i)$ is the length of the full component T^i and $D(T^i)$ is the length of the longest path in T^i.

β-convexity of Steiner tree approximation has been introduced in Ref. [8]. A Steiner tree heuristic A is called a *β-convex α-approximation Steiner tree algorithm* if there exist an integer m and nonnegative real numbers λ_i, $i = 2, \ldots, m$, with $\beta = \sum_{i=2}^m \lambda_i$ and $\alpha = \sum_{i=2}^m \lambda_i \rho_i$ such that the length of the tree computed by A, $l(A)$, is upper bounded by

$$l(A) \leq \sum_{i=2}^{m} \lambda_i \, opt_i$$

where opt_i is the length of the optimal i-restricted Steiner tree.

The MST-algorithm [17] is 1-convex 2-approximation since its output is the optimal 2-restricted Steiner tree of length opt_2. Every k-restricted approximation algorithm from Ref. [16] is 1-convex—the sum of coefficients in the approximation ratio always equals to 1, for example, for $k = 3$, it is 1-convex 11/6-approximation algorithm since the output tree is bounded by $\frac{1}{2}opt_2 + \frac{1}{2}opt_3$. The output tree for Polynomial Time Approximation Scheme (PTAS) [15] converges to the optimal 3-restricted Steiner tree and has length $(1+\epsilon)opt_3$, therefore, it is $(1+\epsilon)$-convex $\frac{5}{3}(1+\epsilon)$-approximation algorithm. The currently best approximation ratio of $1 + \frac{\sqrt{3}}{2}$ is achieved by the heuristic in Ref. [14] which is not known to be β-convex for any value of β.

Given a β-convex α-approximation algorithm A, it follows from Theorem 71.1 that

$$l(A) \leq \sum_i \lambda_i \, opt_i \leq \sum_i \lambda_i \rho_i (opt - D) + \beta D = \alpha(opt - D) + \beta D \tag{71.1}$$

Let OPT be the optimum cost QoSMT tree T, and let t_i be the length of rate r_i edges in T. Then,

$$cost(OPT) = \sum_{i=1}^{N} r_i t_i$$

Let OPT_k be the subtree of the optimal QoS multicast tree OPT induced by edges of rate r_i, $i \geq k$. The tree OPT_k spans the source s and all nodes of rate r_k and, therefore, an optimal Steiner tree connecting s and rate-r_k nodes cannot be longer than

$$l(OPT_k) = \sum_{i=k}^{N} t_i$$

The main idea of the QoSMT algorithms in Ref. [8] is to reuse connections for the higher rate nodes when connecting lower rate nodes. When connecting nodes of rate r_k, we collapse nodes of rate strictly higher than r_k into the source s thus allowing to reuse higher rate connections for free. Let T_k be an approximate Steiner tree connecting the source s and all nodes of rate r_k after collapsing all nodes of rate strictly higher than r_k into the source s and treating all nodes of rate lower than r_k as Steiner points. If we apply an α-approximation Steiner tree algorithm for finding T_k, then the resulted length can be bounded as follows

$$l(T_k) \leq \alpha l(OPT_k) = \alpha t_k + \alpha t_{k+1} + \cdots + \alpha t_N$$

The following lemma shows that if the tree T_k is obtained using β-convex α-approximation Steiner tree algorithm, then a tighter upper bound on the length of T_k holds.

Lemma 71.1 **(Karpinski et al. [8])**

Given an instance of the QoSMT problem, the cost of the tree T_k computed by a β-convex α-approximation Steiner tree algorithm is at most

$$cost(T_k) \leq \alpha r_k t_k + \beta(r_k t_{k+1} + r_k t_{k+2} + \cdots + r_k t_N)$$

Proof

Let OPT_k be the subtree of the optimal QoS multicast tree OPT induced by edges of rate r_i, $i \geq k$. By duplicating nodes and introducing zero length edges, it can be assumed that OPT_{k+1} is a complete binary tree with the set of leaves consisting of the source s and all nodes of rate at least r_{k+1}. The edges of rate r_k form subtrees attached to the tree OPT_{k+1} connecting rate r_k nodes to OPT_{k+1} (see Figure 71.3[a]).

Note that edges of any binary tree T can be partitioned into the edge-disjoint paths connecting internal nodes with leaves as follows. Each internal node v (including the degree-2 root) is split into two nodes v_1 and v_2, such that v_1 becomes a leaf incident to one of the downstream edges and v_2 becomes a degree-2

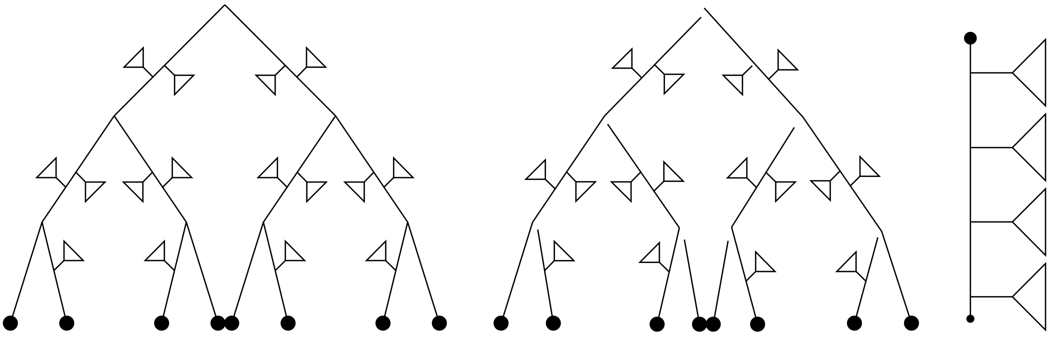


FIGURE 71.3 (a) The subtree OPT_k of the optimal QoS Multicast tree OPT induced by edges of rate r_i, $i \geq k$. Edges of rate greater than r_k (shown as solid lines) form a Steiner tree for $s \cup S_{k+1} \cup \cdots S_N$ (filled circles); attached triangles represent edges of rate r_k. (b) Partition of OPT_k into edge-disjoint connected components OPT_k^i, each containing a single terminal of rate r_i, $i > k$. (c) A connected component OPT_k^i, which consists of a path D_k^i containing all edges of rate r_i, $i > k$, and attached Steiner trees containing edges of rate r_k.

node (or a leaf if v is the root) incident to an edge connecting v to its parent (if v is not the root) and another downstream edge. Since each node is incident to a downstream edge, each resulted connected component will be a path containing exactly one leaf of T connected to an internal node of T.

The binary tree OPT_{k+1} broken into edge-disjoint paths described above along with all nodes of rate r_k that attached to them is shown on Figure 71.3(b). A resulted connected component OPT_k^i consisting of a path $D_k^i = OPT_k^i \cap OPT_{k+1}$ and attached Steiner trees with edges of rate r_k is shown on Figure 71.3(c). Note that the total length of the paths D_k^i is $l(OPT_{k+1}) = t_{k+1} + t_{k+2} + \cdots + t_N$. By Theorem 71.1, decomposing the tree T_k along these full components OPT_k^i results in the following upper bound:

$$\begin{aligned} l(T_k) &\leq \sum_i \left[\alpha\left(l\left(OPT_k^i\right) - D_k^i\right) + \beta D_k^i\right] \\ &= \alpha t_k + \beta(t_{k+1} + t_{k+2} + \cdots + t_N) \end{aligned}$$

The lemma follows by multiplying the last inequality by r_k. □

71.2.4 β-Convex Approximation for QoSMT with Two Rates

In practice, it is often the case that only few distinct rates are requested by the terminals. This is why the QoS problem with two or three rates has a long history [7,10–12]. The previously best results of Refs. [12] have produced algorithms with approximation factor equal to 2.667 (provided that the MST heuristic is used to compute Steiner trees).

In this section approximation factors for the QoSMT problem with two nonzero rates are derived for the balancing algorithm based on β-convex Steiner tree approximation (see Figure 71.4) [8].

Recall that an edge e has rate r_i if the largest node rate in the component of $T - \{e\}$ that does not contain the source is r_i. Let the optimal Steiner tree in G have cost $opt = r_1 t_1 + r_2 t_2$, with t_1 being the total length of the edges of rate r_1 and t_2 being the total length of the edges of rate r_2. The algorithm in Figure 71.4 uses as subroutines two Steiner tree algorithms: an algorithm A_1 with an approximation ratio of α_1, and a β-convex algorithm A_2 with an approximation ratio of α_2. It outputs the minimum cost Steiner tree between the tree $ST1$ obtained by running A_1 with a set of terminals containing the source and the nodes with both high and low nonzero rate, and the tree $ST2$ obtained by running A_1 with a set of terminals containing the source and all high rate nodes, contracting the resulting tree into the source, and running A_2 with a set of terminals containing the contracted source and the low rate nodes.

Input: Graph $G = (V, E, l)$ with two nonzero rates $r_1 < r_2$, source s, terminal sets S_1 of rate r_1 and S_2 of rate r_2, Steiner tree α_1-approximation algorithm A_1 and a β-convex α_2-approximation algorithm A_2.

Output: Low cost QoSMT spanning all terminals.

1. Compute an approximate Steiner tree $ST1$ for $s \cup S_1 \cup S_2$ using algorithm A_1.
2. Compute an approximate Steiner tree T_2 for $s \cup S_2$ (treating all other points as Steiner points) using algorithm A_1. Next, contract T_2 into the source s and compute the approximate Steiner tree T_1 for s and remaining rate r_1 points using algorithm A_2. Let $ST2$ be $T_1 \cup T_2$.
3. Output the minimum cost tree among $ST1$and $ST2$.

FIGURE 71.4 QoSMT approximation algorithm for two nonzero rates.

Theorem 71.2 (Karpinski et al. [8])

The algorithm in Figure 71.4 has an approximation ratio of

$$\max\left\{\alpha_2, \quad \max_r \ \alpha_1 \frac{\alpha_1 - \alpha_2 r + \beta r}{\alpha_1 - \alpha_2 r + \beta r^2}\right\}$$

Proof

The cost of $ST1$ is bounded by $cost(ST1) \leq \alpha_1 r_2 (t_1 + t_2)$. To obtain a bound on the cost of $ST2$ note that $cost(T_2) \leq \alpha_1 r_2 t_2$, and that, by Lemma 71.1, $cost(T_1) \leq \alpha_2 r_1 t_1 + \beta r_1 t_2$.

Thus, the following two bounds for the costs of $ST1$ and $ST2$ follow:

$$\begin{aligned} cost(ST1) &\leq \alpha_1 r_2 t_1 + \alpha_1 r_2 t_2 \\ cost(ST2) &\leq \alpha_1 r_2 t_2 + \alpha_2 r_1 t_1 + \beta r_1 t_2 \end{aligned}$$

Let us distinguish the following two cases: □

Case 1

Let $\beta r_1 \leq (\alpha_2 - \alpha_1) r_2$. Then

$$\begin{aligned} cost(ST2) &\leq \alpha_1 r_2 t_2 + \alpha_2 r_1 t_1 + \beta r_1 t_2 \\ &\leq \alpha_1 r_2 t_2 + \alpha_2 r_1 t_1 + (\alpha_2 - \alpha_1) r_2 t_2 \\ &\leq \alpha_2 (r_2 t_2 + r_1 t_1) \\ &= \alpha_2 opt \end{aligned}$$

Case 2

Let $\beta r_1 > (\alpha_2 - \alpha_1) r_2$. Then the following two values are positive:

$$\begin{aligned} x_1 &= \frac{r_1}{\alpha_1 r_2} (\beta r_1 - (\alpha_2 - \alpha_1) r_2) \\ x_2 &= r_2 - r_1 \end{aligned}$$

The following linear combination will be bounded:

$$\begin{aligned} x_1 cost(ST1) + x_2 cost(ST2) &= \frac{r_1(\beta r_1 - (\alpha_2 - \alpha_1) r_2)}{\alpha_1 r_2} cost(ST1) + (r_2 - r_1) cost(ST2) \\ &\leq r_1(\beta r_1 - (\alpha_2 - \alpha_1) r_2)(t_1 + t_2) + (r_2 - r_1)(\alpha_1 r_2 t_2 + \alpha_2 r_1 t_1 + \beta r_1 t_2) \\ &= \left((\beta - \alpha_2) r_1^2 + r_1 r_2 \alpha_1\right) t_1 + \left((\beta - \alpha_2) r_1 r_2 + r_2^2 \alpha_1\right) t_2 \\ &= ((\beta - \alpha_2) r_1 + r_2 \alpha_1)(r_1 t_1 + r_2 t_2) \\ &\leq (\beta r_1 + \alpha_1 r_2 - \alpha_2 r_1) opt \qquad (71.2) \end{aligned}$$

Let *Approx* be the cost of the tree produced by the approximation algorithm. The Inequality (71.2) implies that

$$\begin{aligned}
Approx &= \min\{cost(ST1), cost(ST2)\} \\
&= \frac{x_1 \min\{cost(ST1), cost(ST2)\} + x_2 \min\{cost(ST1), cost(ST2)\}}{x_1 + x_2} \\
&\leq \frac{x_1 cost(ST1) + x_2 cost(ST2)}{x_1 + x_2} \\
&\leq \frac{\beta r_1 + \alpha_1 r_2 - \alpha_2 r_1}{\frac{r_1}{\alpha_1 r_2}(\beta r_1 - (\alpha_2 - \alpha_1) r_2) + r_2 - r_1} opt \\
&\leq \alpha_1 \frac{\beta r_1 r_2 + \alpha_1 r_2^2 - \alpha_2 r_1 r_2}{\beta r_1^2 - (\alpha_2 - \alpha_1) r_2 r_1 + \alpha_1 r_2^2 - \alpha_1 r_1 r_2} opt \\
&\leq \alpha_1 \frac{\alpha_1 - \alpha_2 r + \beta r}{\alpha_1 - \alpha_2 r + \beta r^2} opt
\end{aligned}$$

where $r = \frac{r_1}{r_2}$.

Summarizing the two cases we obtain that *Approx* is at most the maximum of two values, $\alpha_2 opt$ and $\alpha_1 \frac{\alpha_1 - \alpha_2 r + \beta r}{\alpha_1 - \alpha_2 r + \beta r^2} opt$, which proves the theorem.

Theorem 71.2 implies numerical bounds on the approximation ratios. Using that $\alpha_1 = 1 + \ln 3/2 + \epsilon$ for the algorithm from Ref. [14], $\alpha_2 = 5/3 + \epsilon$ for the algorithm from Ref. [15], $\alpha_1 = \alpha_2 = 11/6$ for the algorithm from Ref. [16], and $\alpha_1 = \alpha_2 = 2$ for the MST heuristic, and $\beta \to 1$ for all of the above algorithms (except for the algorithm from Ref. [14]), we maximize the expression in Theorem 71.2 to obtain the following theorem.

Theorem 71.3 (Karpinski et al. [8])

If the algorithm from Ref. [14] is used as A_1 and the algorithm from Ref. [15] is used as A_2, then the approximation ratio of the QoSMT algorithm in Figure 71.4 is $1.960 + \epsilon$. *If the algorithm from Ref. [15] is used in place of both A_1 and A_2, then the approximation ratio is* $2.059 + \epsilon$. *If the algorithm from Ref. [16] is used in place of both A_1 and A_2, then the ratio is 2.237. If the MST heuristic is used in place of both A_1 and A_2, then the ratio is 2.414.*

71.2.5 β-Convex Approximation for QoSMT with Unbounded Number of Rates

In this section, we describe and prove the performance ratios of β-convex approximation algorithms for the case of the QoSMT problem with arbitrarily many nonzero rates $r_1 < r_2 < \cdots < r_N$ [8]. The algorithm (see Figure 71.5) is a modification of the algorithm in Ref. [6]. As in Ref. [6], node rates are rounded up to the closest power of some number a starting with a^y, where y is picked uniformly at random between 0 and 1. In other words, the given rates are replaced with numbers from the set $\{a^y, a^{y+1}, a^{y+2}, \ldots\}$. The major difference is that each approximate Steiner tree, T_k, constructed over nodes of rounded rate a^{y+k} is contracted in increasing order of k instead of simply taking union of T_k's according to Ref. [6]. This allows contracted edges to be reused at zero cost by Steiner trees connecting lower rate nodes. The following analysis from Ref. [8] of this improvement shows that it decreases the approximation ratio from 4.211 to 3.802.

Let T_{opt} be the optimal QoS Multicast tree, and let t_i be the total length of the edges of T_{opt} with rates rounded to a^{y+i}. First, we prove the following "randomized doubling" lemma corresponding to Lemma 4 from Ref. [6].

```
Input: Graph $G = (V, E, l)$, source $s$, sets $S_i$ of terminals with rate $r_i$, positive number $a$, and
$\alpha$-approximation $\beta$-convex Steiner tree algorithm.
Output: Low cost QoSMT spanning all terminals.
________________________________________________________________________________

1. Pick $y$ uniformly at random between 0 and 1. Round up each rate to the closest power of some number $a$
   starting with $a^y$, that is, round up to numbers in the set $\{a^y, a^{y+1}, a^{y+2}, \ldots\}$. Form new terminal sets $S'_i$ which
   are unions of terminal sets with rates rounded to the same number $r'_i$.
2. $T \leftarrow \emptyset$.
3. For each nonzero rounded rate $r'_i$, in decreasing order, do:
     Find an $\alpha$-approximate Steiner tree $T_i$ spanning $s \cup S'_i$
     $T \leftarrow T \cup T_i$
     Contract $T_i$ into source $s$
4. Output $T$.
```

FIGURE 71.5 Approximation algorithm for multirate QoSMT.

Lemma 71.2 (Karpinski et al. [8])

Let S be the cost of T_{opt} after rounding node rates as in Figure 71.5, that is, $S = \sum_{i=0}^{n} t_i a^{y+i}$. Then,

$$S \leq \frac{a-1}{\ln(a)} cost(T_{opt})$$

Proof

First, note that an edge e used at rate r in T_{opt} will be used at the rate a^{y+m}, where m is the smallest integer i such that a^{y+i} is no less than r. Indeed, e is used at rate r in T_{opt} if and only if the maximum rate of a node connecting to the source via e is r, and every such node will be rounded to a^{y+m}. Next, let $r = a^{x+m}$. If $x \leq y$, then the rounded up cost is a^{y-x} times the original cost; otherwise, if $x > y$, is a^{y+1-x} times the original cost. Hence, the expected factor by which the cost of each edge increases is

$$\int_0^x a^{y+x-1} dy + \int_x^1 a^{y-x}\, dy = \frac{a-1}{\ln a}$$

By linearity of expectation, the expected cost after rounding of T_{opt} is

$$S \leq \frac{a-1}{\ln a} cost(T_{opt})$$

□

Theorem 71.4 (Karpinski et al. [8])

The algorithm given in Figure 71.5 has an approximation ratio of

$$\min_a \left(\alpha \frac{a}{\ln a} - (\alpha - \beta) \frac{1}{\ln a} \right)$$

Proof **(Sketch)**
Let *Approx* be the cost of the tree returned by the algorithm in Figure 71.5, and $Approx_k$ be the cost of the tree T_k constructed by the algorithm when considering rate r_k. Then, by Lemma 71.1,

$$Approx_k \leq \alpha a^{y+k} t_k + \beta a^{y+k+1} t_{k+1} + \beta a^{y+k+2} t_{k+2} + \cdots + \beta a^{y+n} t_n$$

Summing up all the $Approx_k$'s (we omit the details), we get an upper bound of

$$\begin{aligned}(\alpha - \beta)S + \beta S\left(1 + \frac{1}{a} + \frac{1}{a^2} + \cdots\right) &\leq (\alpha - \beta)\frac{a-1}{\ln a} cost(T_{opt}) + \beta\frac{a}{\ln a} cost(T_{opt}) \\ &= \left(\alpha\frac{a}{\ln a} - (\alpha - \beta)\frac{1}{\ln a}\right) cost(T_{opt})\end{aligned}$$

where the last inequality follows from Lemma 71.2. □

Note that the corresponding approximation ratio in Ref. [6] is larger and equals $\alpha\frac{a}{\ln a}$ attaining minimum for $a = e$. The minimum of the approximation ratio in Theorem 71.4 can be obtained numerically—it is equal to 3.802, 4.059, respectively 4.311, when the β-convex α-approximation Steiner tree algorithm used in Figure 71.4 is the algorithm in Refs. [15,16], respectively, the MST heuristic. Finally, the algorithm in Figure 71.5 can be derandomized using the same techniques as in Ref. [6].

71.3 Primal-Dual Approach to the QoSMT Problem

In this section, we discuss several primal-dual heuristics for the QoSMT problem due to Călinescu et al. [13]. A simpler integer linear program and two primal-dual algorithms based on it are discussed in Sections 71.3.1 and 71.3.2. A tighter integer linear program and an associated 4.311-approximation primal-dual algorithm are then described in Section 71.3.3.

71.3.1 A Simple Integer Linear Program (ILP) Formulation

The QoSMT problem can be formulated as an integer program as follows. Consider a network $G = (V, E, length, rate)$ with a source node s and a set of terminal nodes. Let $r_1 < r_2 < \cdots < r_N$ be all rate values assigned to the terminals. To simplify our notation we assume that every node has an extra rate $r_0 = 0$ (i.e., assign rate r_0 to each nonterminal node). As before the source s has the highest rate. Construct a new network $G' = (V, E', cost, rate)$ by replacing each edge e of G with k edges $(e, r_1), (e, r_2), \ldots, (e, r_k)$ and setting $cost((e, r_i)) = r_i \cdot length(e)$.

Let $x_{(e,r)}$ be a Boolean variable denoting whether edge e is used at rate r in an optimum tree. The QoS Steiner tree problem can be formulated as

$$\min \sum_{(e,r)\in E'} x_{(e,r)} \cdot r \cdot length(e) \tag{71.3}$$

$$\text{s.t.} \sum_{\substack{(e,r)\in\delta(C)\\ r\geq r_C}} x_{(e,r)} \geq 1, \quad \forall C \subseteq V\backslash\{s\} \tag{71.4}$$

$$x_{(e,r)} \in \{0, 1\} \tag{71.5}$$

where $\delta(C)$ denotes the set of edges with exactly one endpoint in C and r_C denotes the maximum rate of a node in C. Note that formula (71.3) gives the cost of an optimal solution, while Eq. (71.4) guarantees that each terminal is connected to the source through a collection of edges of rate no less than its rate.

After relaxing the integrality Constraint (71.5), the dual linear program can be written as follows. For each (e, r), $C^*(e, r)$ is defined as $\{C \in V\backslash\{s\} : (e, r) \in \delta(C), r \geq r_C\}$. In words, $C^*(e, r)$ is the set of subsets C of $V\backslash\{s\}$ such that (e, r) has at least one endpoint in C and r is at least as large as r_C. Using this definition, the dual is as follows:

$$
\begin{aligned}
\max \quad & \sum_{C} y_C \\
\text{s.t.} \quad & \sum_{C \in C^*(e,r)} y_C \le r \cdot length(e), \quad \forall (e, r) \\
& y_C \ge 0
\end{aligned}
$$

71.3.2 Two Primal-Dual Methods for the Simple ILP Formulation

The primal-dual framework applied to network design problems usually grows uniformly the dual variables associated to the "active" components of the current forest [22]. This approach fails to take into account the different rates of different nodes in the QoSMT problem. The *Naive Primal-Dual algorithm* [13] (see Figure 71.6) takes into account different rates by varying the speed at which each component grows. While the simulations in the ensuing sections show that this is a good method in practice, the solution it produces on some graphs may be very large compared to the optimal solution, as shown by the following example with two rates.

Consider two nodes of rate 1 connected by an edge of length 1 (see Figure 71.7). There is an arc between these two nodes, and on this arc there is a chain of nodes of rate ϵ. Each two consecutive nodes in the chain are at a distance δ from each other, where $\delta < 1$. Each extreme node in the chain is at a distance $\delta/2$ of its neighboring rate-1 node.

The Naive Primal-Dual applied to this graph connects the rate-ϵ nodes first, since $\frac{\delta}{2} < \frac{1}{2}$. So, the algorithm connects the rate-1 nodes via the rate-ϵ nodes, and not via the direct edge connecting them. Thus, the Naive Primal-Dual can make arbitrarily large errors (just take an arbitrarily long chain).

Input: A graph $G = (V, E, length, rate)$ with a source s in V and a collection of terminals $S \subseteq V$.

Output: A QoSMT spanning the source and the terminals.

1. Start from the spanning forest of G with no edges.
2. Grow y_C with speed r_C for each "active" component C of the current forest. (A component C is *inactive* if it contains s and all vertices of rate r_C.)
3. Stop growing once the dual inequality for a pair (e, r) becomes tight, with e connecting two distinct components of the forest.
4. Add e to the forest, collapsing the two components.
5. Terminate when there is no active component left.
6. Keep an edge of the resulting tree at the minimum needed rate.

FIGURE 71.6 The Naive Primal-Dual algorithm for the QoSMT problem.

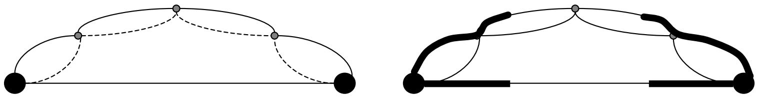


FIGURE 71.7 The Restarting Primal-Dual avoids the mistake of the Naive Primal-Dual. Part (a) shows duplication of the edges. Part (b) shows the components growing along the respective edges.

Input: A Graph $G' = (V, E, cost, rate)$ with source s, and a collection of terminals S.

Output: A QoSMT spanning the source and the terminal.

1. Grow each active C_{r_i} with speed r_i along incident edges (e, r_j), $j \leq i$, picking edges, which become tight.
2. Continue this process until there is no active component of rate r_k.
3. Remove all edges, that are not necessary for maintaining connectivity of nodes of rate r_k.
4. Accept (keep in the solution) and contract all edges of C_{r_k} (i.e., set their length/cost to 0).
5. Restart the algorithm with the new graph.

FIGURE 71.8 The Restarting Primal-Dual algorithm for the QoSMT problem.

An improved *Restarting Primal-Dual algorithm* [13] is given in Figure 71.8. One can easily see that this is a primal-dual algorithm. Indeed, each addition of an edge to the current solution is the result of growing dual variables. Moreover, since the feasibility requirement for edge a is $\Sigma_{a\in\delta(C)} y_C \leq r \cdot length(a)$, this addition preserves the feasibility of the dual solution. The algorithm maintains forests F^{r_i} given by the edges picked at rate r_i, and the connected components of F^{r_i}, seen as sets of vertices, are denoted in the algorithm by C_{r_i}. Such a component is *active* if $r_{C_{r_i}} = r_i$, and C_{r_i} is disjoint from components of higher rate.

The Restarting Primal-Dual algorithm avoids the mistake made by the Naive Primal-Dual algorithm on the frame example in Figure 71.7(a). Then, at time $\frac{\delta}{2}$ the rate-ϵ nodes become connected. This means that $\delta(1-\epsilon)$ of each rate-1 edge between the ϵ-rate nodes is not covered. Meanwhile, the rate-1 nodes are growing on the respective edges as shown in Figure 71.7(b).

Let us assume that the Restarting Primal-Dual algorithm uses the chain of rate-ϵ nodes to connect the two rate-1 nodes instead of the direct edge. This would imply that it takes less time to cover the chain, that is, $\frac{1}{2}\delta(1-\epsilon)n \leq \frac{1}{2} - \frac{\delta}{2}$, where n is the number of rate-ϵ nodes. When ϵ is small, then $n\delta \leq 1$, so if the Restarting Primal-Dual algorithm uses the chain then it is correct to do so.

71.3.3 Primal-Dual 4.311-Approximation Algorithm

A constant-factor primal-dual approximation algorithm is obtained in Ref. [13] based on an enhanced integer linear programming formulation of the QoSMT problem. The enhanced formulation takes into account the fact that if a set $C \subset V\backslash\{s\}$ is connected to the source with edges of rate $r' > r_C$, then there should be at least *two* edges of rate r' with exactly one endpoint in C. The integer program is

$$\begin{aligned} \min \quad & \sum_{(e,r)\in E'} x_{(e,r)} \cdot r \cdot length(e) \\ \text{s.t.} \quad & \sum_{\substack{e\in\delta(C)\\ r=r_C}} x_{(e,r)} + \frac{1}{2}\sum_{\substack{e\in\delta(C)\\ r>r_C}} x_{(e,r)} \geq 1, \quad \forall C \subseteq V \setminus \{s\} \\ & x_{(e,r)} \in \{0, 1\} \end{aligned}$$

The corresponding dual of the LP relaxation is

$$\begin{aligned} \max \quad & \sum_{C\subseteq V\backslash\{s\}} y_C \\ \text{s.t.} \quad & \sum_{\substack{C: e\in\delta(C)\\ r_C=r}} y_C + \frac{1}{2}\sum_{\substack{C: e\in\delta(C)\\ r_C<r}} y_C \leq r \cdot length(e) \\ & y_C \geq 0 \end{aligned} \tag{71.6}$$

Input: A graph $G = (V, E, length, rate)$ with source s in V and a collection of terminals $S \subseteq V$.

Output: A QoSMT spanning the source and the terminal.

```
1. For each $r = r_1, r_2, \ldots, r_k$, execute steps 2–6.
2. Start from the spanning forest $F^r$ of $G$ with no edges.
3. Grow $y_C$ uniformly for each $r$-component $C$ of the current forest $F^r$.
4. Stop growing once the dual inequality for a pair $(e, r)$ becomes tight, with $e$ connecting two distinct
   components of $F^r$.
5. Add $(e, r)$ to $F^r$, collapsing two of its components.
6. Terminate when there is no $r$-component of $F^r$ left.
7. Traversing the list of picked edges in reverse order, remove an edge $(e, r)$ from $F^r$ if after $(e, r)$'s removal
   the set of edges picked form a feasible tree.
```

FIGURE 71.9 The 4.311-approximation algorithm for QoSMT problem.

The core algorithm presented in Figure 71.9 is preprocessed with random bucketing of rates as in Section 71.2.5 (see also step 1 in Figure 71.5). Let a be a real (to be picked later) and γ be a real picked uniformly at random from the interval $[0\,..\,1]$. Every node of rate r is replaced by a node of rate $a^{\gamma+j}$, where j is the integer satisfying $a^{\gamma+j-1} < r \leq a^{\gamma+j}$.

The primal-dual part follows the classical framework [22], and works in stages starting from the lower rate to the highest. During the execution of the algorithm, edges are picked at a certain rate (in other words, $x_{(e,r)}$ is set to 1) one by one. Before executing step 3 at rate r for the ith time, the set of edges picked at rate r by the algorithm forms a forest F_i^r. (An edge can be picked at several rates, but it is kept in at most one such rate in the final solution because of the reverse delete step.) A component C of F_i^r is called an *r-component* if $r_C = r$.

Using constraint (71.6), it follows by induction on j that, for an edge e and a rate $a^{\gamma+j}$, we have

$$\begin{aligned}\sum_{\substack{C\,:e\in\delta(C)\\ r_C\leq a^{\gamma+j}}} y_C &\leq length(e)a^{\gamma+j}\sum_{i=0}^{j}\left(\frac{1}{2a}\right)^i\\ &\leq length(e)a^{\gamma+j}\frac{2a}{2a-1}\end{aligned}$$

For an edge picked by the algorithm at rate r, Constraint (71.6) is tight and therefore,

$$\sum_{\substack{C\,:e\in\delta(C)\\ r_C\leq a^{\gamma+j}}} y_C \geq length(e)\frac{2a-2}{2a-1}a^{\gamma+j} \tag{71.7}$$

Exactly as in Ref. [22], the number of edges of rate r in the final solution which cross the active r-components at some moment (an edge being counted twice if it crosses two r-components) is at most twice the number of active r-components. Eq. (71.7), and exactly the same argument as in Theorem 4.2 of Ref. [22], imply that the cost of the solution of the algorithm is bounded by $(2(2a-1)/(2a-2))$ $\sum y_C \leq ((2a-1)/(a-1))\ opt$, as any feasible solution for the dual linear program has value at most the value of any feasible solution of the primal.

The same argument as in Section 71.2.5 shows that the approximation ratio of the algorithm above is $(2a-1)/\ln a$. Numerical picking the same best value for a as in Section 71.2.5 implies

Theorem 71.5 (Călinescu et al. [13])

The output cost of the algorithm on Figure 71.9 is at most 4.311 *times the optimum cost.*

71.4 Experimental Study

In this section, we report experimental results with several QoSMT heuristics: Maxemchuk's [5], binary rounding [6], naive primal-dual, and restarting primal-dual algorithms. The heuristics were implemented in C++ and compiled using gpp with `-O2` optimization, and run on a Sun workstation Ultra-60. The experiments were run on random testcases generated using GT-ITM generator [23], which is used for modeling Internet networks [24]. Table 71.3 gives a comparison of the performance of of the aforementioned algorithms. The experiments were conducted in the presence of no Steiner nodes, respectively 50% Steiner nodes. Moreover, both arithmetic and geometric distributions of rates were tested.

TABLE 71.3 Cost Improvement Over Maxemchuck's Algorithm (%) and CPU Seconds for Binary Rounding and Two Primal-Dual Algorithms (Averages Over Ten Testcases)

50% Steiner Nodes, Geometric Progression Rates								
		Maxemchuk's	Binary rounding		Naive-PD		Restart-PD	
R	*N*	CPU	G(%)	CPU	G(%)	CPU	G(%)	CPU
1	200	0.017	0.00	0.017	−0.01	0.544	−0.01	0.325
1	300	0.050	0.00	0.052	0.04	1.372	0.04	0.946
2	200	0.027	0.00	0.026	0.43	1.271	1.03	1.125
2	300	0.070	0.00	0.072	0.93	4.573	2.17	3.747
5	200	0.044	0.00	0.044	−2.13	1.490	1.30	5.321
5	300	0.123	0.00	0.120	−0.91	5.221	1.10	16.798
10	200	0.065	0.00	0.068	−2.53	1.636	0.66	17.848
10	300	0.180	0.00	0.176	−2.61	6.582	0.24	107.125
50% Steiner Nodes, Arithmetic Progression Rates								
1	200	0.016	0.00	0.017	−0.01	0.541	−0.01	0.327
1	300	0.052	0.00	0.051	0.04	1.370	0.04	0.946
2	200	0.027	0.00	0.023	−0.69	1.373	−0.00	1.136
2	300	0.071	0.00	0.070	−0.32	4.491	0.24	3.773
5	200	0.043	−0.01	0.040	1.70	1.564	2.66	5.256
5	300	0.123	−0.10	0.107	1.92	5.392	4.19	17.271
10	200	0.067	1.79	0.043	4.25	1.556	6.11	16.856
10	300	0.181	2.36	0.126	3.38	5.444	5.73	92.575
0% Steiner Nodes, Geometric Progression Rates								
1	100	0.002	0.00	0.002	0.00	0.052	0.00	0.077
1	200	0.028	0.00	0.028	0.00	0.251	0.00	0.465
2	100	0.007	0.00	0.007	1.21	0.088	1.69	0.185
2	200	0.038	0.00	0.033	2.14	0.698	2.31	1.517
5	100	0.012	0.00	0.013	1.24	0.120	2.82	0.665
5	200	0.059	0.00	0.056	−0.25	1.296	1.70	6.314
10	100	0.019	0.00	0.018	−0.68	0.133	1.63	1.953
10	200	0.090	0.00	0.091	−1.97	1.466	0.73	20.525
0% Steiner Nodes, Arithmetic Progression Rates								
1	100	0.005	0.00	0.005	0.00	0.054	0.00	0.078
1	200	0.026	0.00	0.026	0.00	0.247	0.00	0.457
2	100	0.005	0.00	0.006	−0.11	0.111	−0.04	0.187
2	200	0.036	0.00	0.034	−0.02	1.078	0.30	1.570
5	100	0.011	−0.17	0.011	3.70	0.114	4.60	0.656
5	200	0.059	−0.15	0.052	3.13	1.235	3.85	5.952
10	100	0.019	2.62	0.012	6.65	0.113	7.12	1.922
10	200	0.091	2.67	0.058	5.83	1.203	6.38	17.689

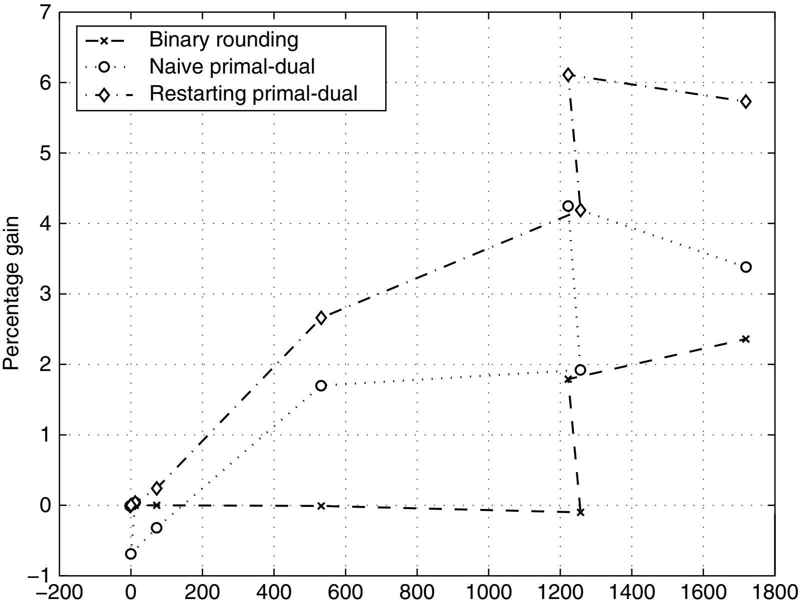


FIGURE 71.10 The gain of several algorithms versus Maxemchuk's algorithm, 50% Steiner nodes.

Table 71.3 gives the results for instances generated using several sets of parameters. The relative solution quality of various heuristics is fairly independent on the class of instances. We note that the Naive Primal-Dual and the Charikar–Naor–Schieber algorithms most often produce comparable results, which are slight improvements over the results produced by Maxemchuk's algorithm. The Restarting Primal-Dual consistently produces solutions of best quality, typically 0.25–6% better than solutions produced by Maxemchuk's algorithm; this, however, occurs at the expense of greater CPU time. We also note that the difference between algorithms increases as the number of rates increases. Figure 71.10 and Figure 71.11 illustrate this observation in a graphical form.

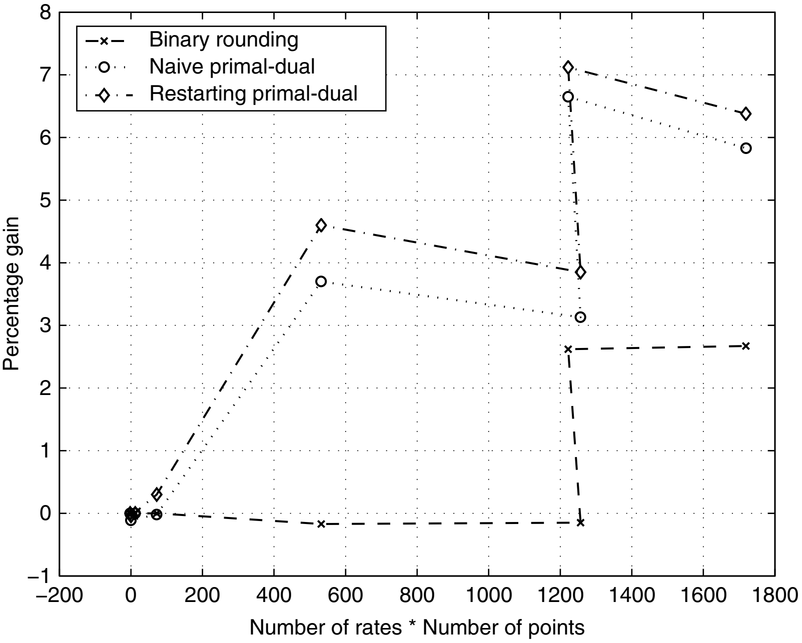


FIGURE 71.11 The gain of several algorithms versus Maxemchuk's algorithm, 0% Steiner nodes.

References

[1] Salama, H. F., Reeves, D. S., and Viniotis, Y., Evaluation of multicast routing algorithms for real-time communication on high-speed networks, *IEEE J. Sel. Areas in Comm.*, 3, 332, 1997.

[2] Ural, H. and Zhu, K., An efficient distributed QoS based multicast routing algorithm, *Proc. Int. Perf., Comput., and Comm. Conf.*, 2002, p. 27.

[3] Bajaj, R., Ravikumar, C. P., and Chandra, S., Distributed delay constrained multicast path setup high speed networks, *Proc. Int. Conf. on High Perf. Comput.*, 1997, p. 438.

[4] Rouskas, R. N. and Baldine, I., Multicast routing with end-to-end delay and delay variation constraints, *IEEE J. Sel. Areas Comm.*, 15, 346, 1997.

[5] Maxemchuk, N., Video distribution on multicast networks, *IEEE J. Sel. Areas in Comm.*, 15, 357, 1997.

[6] Charikar, M., Naor, J., and Schieber, B., Resource optimization in QoS multicast routing of real-time multimedia, *IEEE/ACM Trans. Networking*, 12, 340, 2004.

[7] Xue, G., Lin, G.-H., and Du, D.-Z., Grade of service Steiner minimum trees in the Euclidean plane, *Algorithmica*, 31, 479, 2001.

[8] Karpinski, M., Măndoiu, I., Olshevsky, A., and Zelikovsky, A., Improved approximation algorithms for the quality of service Steiner tree problem, *Algorithmica*, 42, 109, 2005.

[9] Current, J. R., Revelle, C. S., and Cohon, J. L., The hierarchical network design problem, *Eur. J. Oper. Res.*, 27, 57, 1986.

[10] Balakrishnan, A., Magnanti, T. L., and Mirchandani, P., Modeling and heuristic worst-case performance analysis of the two-level network design problem, *Manage. Sci.*, 40, 846, 1994.

[11] Balakrishnan, A., Magnanti, T. L., and Mirchandani, P., Heuristics, LPs, and trees on trees: network design analyses, *Oper. Res.*, 44, 478, 1996.

[12] Mirchandani, P., The multi-tier tree problem, *INFORMS J. Comput.*, 8, 202, 1996.

[13] Călinescu, G., Fernandes, C., Măndoiu, I., Olshevsky, A., Yang, K., and Zelikovsky, A., Primal-dual algorithms for QoS multimedia multicast, *Proc. IEEE GLOBECOM*, 2003, p. 3631.

[14] Robins, G. and Zelikovsky, A., Tighter bounds for graph Steiner tree approximation, *SIAM J. Disc. Math.*, 19, 122, 2005.

[15] Promel, H. and Steger, A., A new approximation algorithm for the Steiner tree problem with performance ratio $\frac{5}{3}$, *J. Algorithms*, 36, 89, 2000.

[16] Berman, P. and Ramaiyer, V., Improved Approximations for the Steiner tree problem, *J. Algorithms*, 17, 381, 1994.

[17] Takahashi, H. and Matsuyama, A., An approximate solution for the Steiner problem in graphs, *Math. Japonica*, 6, 573, 1980.

[18] Zelikovsky, A., An 11/6-approximation algorithm for the network Steiner problem, *Algorithmica*, 9, 463, 1993.

[19] Zelikovsky, A., A faster approximation algorithm for the Steiner tree problem in graphs, *Inf. Proc. Lett.*, 46, 79, 1993.

[20] Mehlhorn, K., A faster approximation algorithm for the Steiner problem in graphs, *Inf. Proc. Lett.*, 27, 125, 1988.

[21] Borchers, A. and Du, D. Z., The k-Steiner ratio in graphs, *SIAM J. Comput.*, 26, 1997, 857.

[22] Goemans, M. and Williamson, D., The primal-dual method for approximation algorithms and its application to network design problems, in *Approximation Algorithms*, Hochbaum, D., Ed., PWS Publishing Company, Boston, MA, 1997, p. 144.

[23] http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz

[24] Zegura, E. W., Calvert, K., and Bhattacharjee, S., How to model an Internetwork? *Proc. INFOCOM*, 1996, p. 594.

[25] Colbourn, C. J. and Xue, G. L., Grade of service Steiner trees in series-parallel networks, in *Advances in Steiner Trees*, Du, D. Z., Smith, J. M., and Rubinstein, J. H., Eds., Kluwer Academic Publishers, 2000, p. 163.

72

Overlay Networks for Peer-to-Peer Networks

Andréa W. Richa
Arizona State University

Christian Scheideler
Technical University of Munich

72.1 Introduction

Every distributed system must be based on some kind of logical interconnection structure, also called an *overlay network,* that allows its sites to exchange information. Once distributed systems become large enough, one has to deal with sites continuously entering and leaving the system, simply because sites may fail and have to be replaced by new sites or because additional resources have to be added to preserve the functionality of the system. Hence, in general, a distributed system supporting any service has to have an overlay network supporting joining, leaving, and routing between the sites, and without a scalable implementation of such a network, the field of scalable distributed systems does not really exist.

Scalability is especially critical for peer-to-peer systems. The basic idea of peer-to-peer systems is to have an open self-organizing system of peers that does not rely on any central server and where peers can join and leave at will. This has the benefit that individuals can cooperate without fees or an investment in additional high-performance hardware. Also, peer-to-peer systems can make use of the tremendous amount of resources (such as computation and storage) that otherwise sit idle on individual computers when they are not in use by their owners.

If we want a scalable peer-to-peer system, then joining, leaving, and routing between the sites should be performed with at most polylogarithmic work in the size of the system. This implies that the maximum degree and the diameter of the overlay network should be at most polylogarithmic as well. The overlay network should also be well connected so that it is robust against faulty peers. The well connectedness of a graph is usually measured by its expansion, which we will formally define later in this chapter. Another important parameter is the stretch factor of an overlay network, which measures by how much the length of a shortest route between two nodes v and w in the overlay network is off from a shortest route from v to w when using the underlying physical network.

To summarize, we are seeking operations JOIN, LEAVE, and ROUTE so that for any sequence of join, leave, and route requests,

- the *work* of executing these requests is as small as possible,
- the *degree, diameter, and stretch factor* of the resulting network are as small as possible, and
- the *expansion* of the resulting network is as large as possible.

Hence, we are dealing with multiobjective optimization problems for which we want to find good approximate solutions. To address these problems, we first introduce some basic notation and techniques for constructing overlay networks (Section 72.2). Afterwards, we start with supervised overlay network designs (i.e., the topology is maintained by a supervisor but routing is done on a peer-to-peer basis), and then we present various decentralized overlay network designs (i.e., the topology is maintained by the peers themselves). For simplicity, we assume that no peer fails and that the peers always execute the operations in a correct and timely manner (though in practice this might not be the case).

72.2 Basic Notation and Techniques

We start with some basic notation. A graph $G = (V, E)$ consists of a node set V and an edge set $E \subseteq V \times V$. We will only consider directed graphs. The *in-degree* of a node is the number of incoming edges, the *out-degree* of a node is the number of outgoing edges, and the *degree* of a node is the number of incoming and outgoing edges. Given two nodes v and w, let $d(v, w)$ denote the length of a shortest directed path from v to w in G. G is strongly connected if $d(v, w)$ is finite for every pair $v, w \in V$. In this case,

$$D = \max_{v,w \in V} d(v, w)$$

is the *diameter* of G. The *expansion* of G is defined as

$$\alpha = \min_{S \subseteq V,\ |S| \leq |V|/2} \frac{|\Gamma(S)|}{|S|}$$

where $\Gamma(S) = \{v \in V \setminus S \mid \exists u \in S : (u, v) \in E\}$ is the neighbor set of S. The following relationship between the expansion and diameter of a graph is easy to show:

Fact 72.1

For any graph G with expansion α, the diameter of G is in $O(\alpha^{-1} \log n)$.

The vast majority of overlay networks for peer-to-peer systems suggested in the literature is based on the concept of virtual space. That is, every site is associated with a point in some space U and connections between sites are established based on rules how to interconnect points in that space. In this case, the following operations need to be implemented:

- JOIN(p): add new peer p to the network by choosing a point in U for it
- LEAVE(p): remove peer p from the network
- ROUTE(m, x): route message m to point x in U

Several virtual space approaches are known. The most influential techniques are the hierarchical decomposition technique, the continuous-discrete technique, and the prefix technique. We will give a general outline of each technique in this section. At the end of this section, we present two important families of graphs that we will use later in this chapter to construct dynamic overlay networks.

72.2.1 The Hierarchical Decomposition Technique

Consider the space $U = [0, 1]^d$ for some fixed $d \geq 1$. The *decomposition tree* $T(U)$ of U is an infinite binary tree in which the root represents U and for every node v representing a subcube U' in U, the

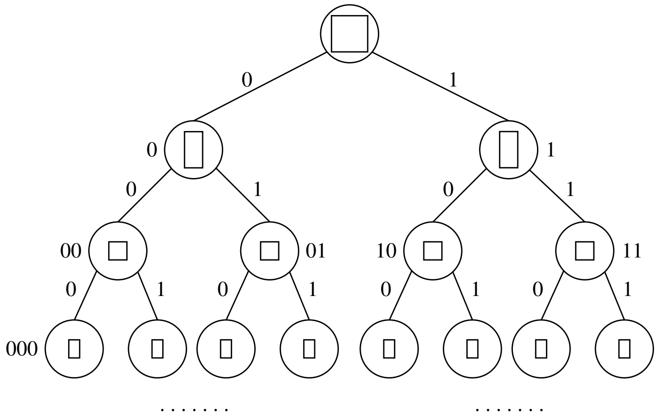


FIGURE 72.1 The decomposition tree for $d = 2$.

children of v represent two subcubes U'' and U''', where U'' and U''' are the results of cutting U' in the middle at the smallest dimension in which U' has a maximum side length. The subcubes U'' and U''' are closed, that is, their intersection gives the cut. Let every edge to a left child in $T(U)$ be labeled with 0 and every edge to a right child in $T(U)$ be labeled with 1. Then the label of a node v, $\ell(v)$, is the sequence of all edge labels encountered when moving along the unique path from the root of $T(U)$ downwards to v. For $d = 2$, the result of this decomposition is shown in Figure 72.1.

The goal is to map the peers to nodes in $T(U)$ so that the following conditions are met:

Condition 72.1

(1) The interiors of the subcubes associated with the (nodes assigned to the) peers are disjoint,
(2) the union of the subcubes of the peers gives the entire set U, and
(3) every peer p with subcube U_p is connected to all peers p' with subcubes $U_{p'}$ that are adjacent to U_p (i.e., $U_p \cap U_{p'}$ is a $d-1$-dimensional subcube).

In the 2-dimensional case, for example, condition (3) means that p and p' share a part of the cut line through their first common ancestor in $T(U)$. It is not difficult to see that the following result is true.

Fact 72.2

Consider the space $U = [0, 1]^d$ for some fixed d and suppose we have n peers. If the peers are associated with nodes that are within k levels of $T(U)$ and Condition 72.1 is satisfied, then the maximum degree of a peer is at most $(2d)2^{k-1}$ and the diameter of the graph is at most $dn^{1/d} + 2(k-1)$.

The diameter of the graph can be as large as $dn^{1/d}$ and therefore too large for a scalable graph if d is fixed, but its degree is small as long as $k = O(\log\log n)$.

An example of a peer-to-peer system using the hierarchical decomposition technique is CAN [1]. In the original CAN construction, a small degree is achieved by giving each peer p a label $\ell(p)$ consisting of a (sufficiently long) random bit string when it joins the system. This bit string is used to route p to the unique peer p' that is reached when traversing the tree $T(U)$ according to $\ell(p)$ (a 0 bit means "go left" and a 1 bit means "go right") until a node v is reached that is associated with a peer. (Such a node must always exist if there is at least one peer in the system and the two rules of assigning peers to nodes in Condition 72.1 are satisfied.) One of p and p' is then placed in the left child of v and the other in the right child of v. Leave operations basically reverse join operations so that Condition 72.1 is maintained. Due to the use of a random bit sequence, one can show that the number of levels the peers are apart is indeed $O(\log\log n)$, as desired, but it can also be as bad as that. Strategies that achieve a more even level balancing were, subsequently, proposed in several papers (see, e.g., Ref. [2] and the references therein).

72.2.2 The Continuous-Discrete Technique

The basic idea underlying the continuous-discrete approach [3] is to define a continuous model of graphs and to apply this continuous model to the discrete setting of a finite set of peers. A well-known peer-to-peer system that uses an approach closely related to the continuous-discrete approach is Chord [4].

Consider the d-dimensional space $U = [0, 1)^d$, and suppose that we have a set F of continuous functions $f_i : U \to U$. Then we define E_F as the set of all pairs $(x, y) \in U^2$ with $y = f_i(x)$ for some i. Given any subset $S \subseteq U$, let $\Gamma(S) = \{y \in U \setminus S \mid \exists x \in S : (x, y) \in E_F\}$. If $\Gamma(S) \neq \emptyset$ for every $S \subset U$, then F is said to be *mixing*. If F does not mix, then there are disconnected areas in U.

Consider now any set of peers V, and let $S(v)$ be the subset in U that has been assigned to peer v. Then the following conditions have to be met:

Condition 72.2

(1) $\cup_v S(v) = U$ and

(2) for every pair of peers v and w it holds that v is connected to w if and only if there are two points $x, y \in U$ with $x \in S(v)$, $y \in S(w)$, and $(x, y) \in E_F$.

Let $G_F(V)$ be the graph resulting from the conditions above. Then the following fact is easy to see:

Fact 72.3

If F is mixing and $\cup_v S(v) = U$, then $G_F(V)$ is strongly connected.

To bound the diameter of $G_F(V)$, we introduce some further notation. For any point x and any $\epsilon \in [0, 1)$, let $B(x, \epsilon)$ denote the d-dimensional ball of volume ϵ centered at x. For any two points x and y in U let $d_n(x, y)$ denote the shortest sequence $(s_1 s_2 s_3 \ldots s_k) \in \mathbb{N}^k$ so that there are two points $x' \in B(x, 1/n)$ and $y' \in B(y, 1/n)$ with $f_{s_1} \circ f_{s_2} \circ \ldots f_{s_k}(x') = y'$. Then we define the diameter of F as

$$D(n) = \max_{x,y \in U} d_n(x, y)$$

Using this definition, it holds

Fact 72.4

If $\cup_v S(v) = U$ and every $S(v)$ contains a ball of volume at least $1/n$ then $G_F(V)$ has a diameter of at most $D(n)$.

Also the expansion of $G_F(V)$ can be bounded with a suitable parameter for F, but it is easier to consider explicit examples here, and therefore we defer a further discussion to Section 72.4.1.

72.2.3 The Prefix Technique

The prefix technique was first presented in Refs. [5,6] and first used in the peer-to-peer world by Pastry [7] and Tapestry [8]. Given a label $\ell = (\ell_1 \ell_2 \ell_3 \ldots)$, let $\text{prefix}_i(\ell) = (\ell_1 \ell_2 \ldots \ell_i)$ for all $i \geq 1$ and $\text{prefix}_0(\ell) = \epsilon$, the empty label.

In the prefix technique, every peer node v is associated with a unique label $\ell(v) = \ell(v)_1 \ldots \ell(v)_k$, where each $\ell(v)_i \in \{0, \ldots, b-1\}$, for some constant $b \geq 2$ and sufficiently large k, and the following condition has to be met concerning connections between the nodes.

Condition 72.3

For every peer v, every digit $\alpha \in \{0, \ldots, b-1\}$, and every $i \geq 0$, v has a link to a peer node w with $\text{prefix}_i(\ell(v)) = \text{prefix}_i(\ell(w))$ *and $\ell(w)_{i+1} = \alpha$, if such a node w exists.*

Since for some values of i and α there can be many nodes w satisfying the condition above, a rule has to be specified which of these nodes w to pick. For example, a peer may connect to the geographically closest peer w, or a peer may connect to a peer w to which it has the best connection. The following result is easy to show:

Fact 72.5

If the maximum length of a node label is L and the node labels are unique, then any rule of choosing a node w as in Condition 72.3 guarantees strong connectivity. Moreover, the maximum out-degree of a node and the diameter of the network are at most L.

However, the in-degree, that is, the number of incoming connections, can be quite high, depending on the rule. An easy strategy guaranteeing polylogarithmic in- and out-degree and logarithmic diameter is that every node chooses a random binary sequence as its label, and a node v connects to the node w among the eligible candidates with the closest distance to v, that is, $|\ell(v) - \ell(w)|$ is minimized. This rule also achieves a good expansion but not a good stretch factor. To address the stretch factor, other rules are necessary that are discussed in Section 72.4.2.

72.2.4 Basic Classes of Graphs

We will apply our basic techniques above to two important classes of graphs: the hypercube and the de Bruijn graph. They are defined as follows.

Definition 72.1

*For any $d \in \mathbb{N}$, the d-*dimensional hypercube *is an undirected graph $G = (V, E)$ with $V = \{0, 1\}^d$ and $E = \{\{v, w\} \mid H(v, w) = 1\}$, where $H(v, w)$ is the Hamming distance between v and w.*

Definition 72.2

*For any $d \in \mathbb{N}$, the d-*dimensional de Bruijn graph *is an undirected graph $G = (V, E)$ with node set $V = \{v \in \{0, 1\}^d\}$ and edge set E that contains all edges $\{v, w\}$ with the property that $w \in \{(x, v_{d-1}, \ldots, v_1) : x \in \{0, 1\}\}$, where $v = (v_{d-1}, \ldots, v_0)$.*

72.3 Supervised Overlay Networks

A *supervised overlay network* is a network formed by a supervisor but in which all other activities can be performed on a peer-to-peer basis without involving the supervisor. It can, therefore, be seen as being between server-based overlay networks and pure peer-to-peer overlay networks. In order for a supervised network to be highly scalable, two central requirements have to be satisfied:

1. The supervisor needs to store at most a polylogarithmic amount of information about the system at any time (e.g., if there are n peers in the system, storing contact information about $O(\log^2 n)$ of these peers would be fine).
2. The supervisor needs at most a constant number of messages to include a new peer into or exclude an old peer from the network.

The second condition makes sure that the work of the supervisor to include or exclude peers from the system is kept at a minimum. First, we present a general strategy of constructing supervised overlay networks, which combines the hierarchical decomposition technique with the continuous-discrete technique and the recursive labeling technique below, and then we give some explicit examples that achieve near-optimal results for the cost of the join, leave, and route operations as well as the degree, diameter, and expansion of the network.

72.3.1 The Recursive Labeling Technique

In the recursive labeling approach, the supervisor assigns a *label* to every peer that wants to join the system. The labels are represented as binary strings and are generated in the following order:

$$0, 1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots$$

Basically, when stripping off the least significant bit, the supervisor is first creating all binary numbers of length 0, then length 1, then length 2, and so on. More formally, consider the mapping $\ell : \mathbb{N}_0 \rightarrow \{0, 1\}^*$ with the property that for every $x \in \mathbb{N}_0$ with binary representation $(x_d \ldots x_0)_2$ (where d is minimum possible):

$$\ell(x) = (x_{d-1} \ldots x_0 x_d)$$

Then ℓ generates the sequence of labels displayed above. In the following, it will also be helpful to view labels as real numbers in $[0, 1)$. Let the function $r : \{0, 1\}^* \rightarrow [0, 1)$ be defined so that for every label $\ell = (\ell_1 \ell_2 \ldots \ell_d) \in \{0, 1\}^*$, $r(\ell) = \sum_{i=1}^{d} \frac{\ell_i}{2^i}$. Then the sequence of labels above translates into

$$0,\ 1/2,\ 1/4,\ 3/4,\ 1/8,\ 3/8,\ 5/8,\ 7/8,\ 1/16,\ 3/16,\ 5/16,\ 7/16,\ 9/16, \ldots$$

Thus, the more labels are used, the more densely the $[0, 1)$ interval will be populated. When using the recursive approach, the supervisor aims to maintain the following condition at any time:

Condition 72.4

The set of labels used by the peers is $\{\ell(0), \ell(1), \ldots, \ell(n-1)\}$, *where n is the current number of peers in the system.*

This condition is preserved when using the following simple strategy:

- Whenever a new peer v joins the system and the current number of peers is n, the supervisor assigns the label $\ell(n)$ to v and increases n by 1.
- Whenever a peer w with label ℓ wants to leave the system, the supervisor asks the peer with currently highest label $\ell(n-1)$ to take over the role of w (and thereby change its label to ℓ) and reduces n by 1.

72.3.2 Putting the Pieces Together

We assume that we have a single supervisor for maintaining the overlay network. In the following, the label assigned to some peer v will be denoted as ℓ_v. Given n peers with unique labels, we define the *predecessor* $\text{pred}(v)$ of peer v as the peer w for which $r(\ell_w)$ is closest from below to $r(\ell_v)$, and we define the *successor* $\text{succ}(v)$ of peer v as the peer w for which $r(\ell_w)$ is closest from above to $r(\ell_v)$ (viewing $[0, 1)$ as a ring in both cases). Given two peers v and w, we define their *distance* as

$$\delta(v, w) = \min\{(1 + r(\ell_v) - r(\ell_w)) \bmod 1,\ (1 + r(\ell_w) - r(\ell_v)) \bmod 1\}$$

To maintain a doubly linked cycle among the peers, we simply have to maintain the following condition:

Condition 72.5

Every peer v in the system is connected to $\text{pred}(v)$ *and* $\text{succ}(v)$.

Now, suppose that the labels of the peers are generated via the recursive strategy above. Then we have the following properties:

Lemma 72.1

Let n be the current number of peers in the system, and let $\bar{n} = 2^{\lfloor \log n \rfloor}$. Then for every peer $v \in V$, $|\ell_v| \leq \lceil \log n \rceil$ and $\delta(v, \text{pred}(v)) \in \{1/(2\bar{n}), 1/\bar{n}\}$.

So the peers are approximately evenly distributed in $[0, 1)$ and the number of bits for storing a label is almost as low as it can be without violating the uniqueness requirement.

Now, recall the hierarchical decomposition approach. The supervisor will assign every peer p to the unique node v in $T(U)$ at level $\log(1/\delta(p, \text{pred}(p)))$ with ℓ_v being equal to ℓ_p (padded with 0's to the right so that $|\ell_v| = |\ell_p|$). As an example, if we have four peers currently in the system, then the mapping of peer labels to node labels is

$$0 \rightarrow 00, \;\; 1 \rightarrow 10, \;\; 01 \rightarrow 01, \;\; 11 \rightarrow 11$$

With this strategy, it follows from Lemma 72.1 that Fact 72.2 applies with $k = 2$.

Consider now any family F of functions acting on some space $U = [0, 1)^d$ and let $C(p)$ be the subcube of the node in $T(U)$ that p has been assigned to. Then the goal of the supervisor is to maintain the following condition at any time:

Condition 72.6

For the current set V of peers in the system it holds that

1. *the set of labels used by the peers is $\{\ell(0), \ell(1), \ldots, \ell(n-1)\}$, where $n = |V|$,*
2. *every peer v in the system is connected to* $\text{pred}(v)$ *and* $\text{succ}(v)$, *and*
3. *there is an edge (v, w) for every pair of peers v and w for which there is an edge $(x, y) \in E_F$ with $x \in C(v)$ and $y \in C(w)$.*

72.3.3 Maintaining Condition 72.6

Next we describe the actions that the supervisor has to perform to maintain Condition 72.6 during a join or leave operation. We start with the following important fact:

Fact 72.6

Whenever a new peer v enters the system, then $\text{pred}(v)$ *has all the connectivity information v needs to satisfy Condition 72.6(3), and whenever an old peer w leaves the system, then it suffices that it transfers all of its connectivity information to* $\text{pred}(w)$ *to maintain Condition 72.6(3).*

The first part of the fact follows from the observation that when v enters the system, then the subcube of $\text{pred}(v)$ splits into two subcubes where one resides at $\text{pred}(v)$ and the other is taken over by v. Hence, if $\text{pred}(v)$ passes all of its connectivity information to v, then v can establish all edges relevant for it according to the continuous-discrete approach. The second part of the fact follows from the observation that the departure of a peer is the reverse of the insertion of a peer.

Thus, if the peers take care of the connections in Condition 72.6(3), the only part that the supervisor has to take care of is maintaining the cycle. For this we require the following condition:

Condition 72.7

At any time, the supervisor stores the contact information of $\text{pred}(v)$, v, $\text{succ}(v)$, *and* $\text{succ}(\text{succ}(v))$, *where v is the peer with label $\ell(n-1)$.*

To satisfy Condition 72.7, the supervisor performs the following actions. If a new peer w joins, then the supervisor

- informs w that $\ell(n)$ is its label, $\text{succ}(v)$ its predecessor, and $\text{succ}(\text{succ}(v))$ its successor,
- informs $\text{succ}(v)$ that w is its new successor,
- informs $\text{succ}(\text{succ}(v))$ that w is its new predecessor,
- asks $\text{succ}(\text{succ}(v))$ to send its successor information to the supervisor, and
- sets $n = n + 1$.

If an old node w leaves and reports ℓ_w, pred(w), and succ(w) to the supervisor (recall that we are assuming graceful departures), then the supervisor

- informs v (the node with label $\ell(n-1)$) that ℓ_w its new label, pred(w) its new predecessor, and succ(w) its new successor,
- informs pred(w) that its new successor is v and succ(w) that its new predecessor is v,
- informs pred(v) that succ(v) is its new successor and succ(v) that pred(v) is its new predecessor,
- asks pred(v) to send its predecessor information to the supervisor and to ask pred(pred(v)) to send its predecessor information to the supervisor, and
- sets $n = n - 1$.

Thus, the supervisor only needs to handle a small constant number of messages for each arrival or departure of a peer, as desired. Next we look at two examples resulting in scalable supervised overlay networks.

72.3.4 Examples

For a supervised hypercubic network, simply select F as the family of functions on $[0, 1)$ with $f_i(x) = x + 1/2^i \pmod 1$ for every $i \geq 1$. Using our framework, this gives an overlay network with degree $O(\log n)$, diameter $O(\log n)$, and expansion $O(1/\sqrt{\log n})$, which matches the properties of ordinary hypercubes.

For a supervised de Bruijn network, simply select F as the family of functions on $[0, 1)$ with $f_0(x) = x/2$ and $f_1(x) = (1 + x)/2$. Using our framework, this gives an overlay network with degree $O(1)$, diameter $O(\log n)$, and expansion $O(1/\log n)$, which matches the properties of ordinary de Bruijn graphs.

In both networks, routing with logarithmic work can be achieved by using the bit adjustment strategy.

72.4 Decentralized Overlay Networks

Next we show that scalable overlay networks can also be maintained without involving a supervisor. We only discuss examples for the latter two basic techniques in Section 72.2 since the hierarchical decomposition technique cannot yield networks of polylogarithmic diameter.

72.4.1 Overlay Networks Based on the Continuous-Discrete Approach

Similar to the supervised approach, we first show how to maintain a hypercubic overlay network, and then we show how to maintain a de Bruijn-based overlay network.

72.4.1.1 Maintaining a Dynamic Hypercube

Let $U = [0, 1)$ and consider the family F of functions on $[0, 1)$ with $f_i(x) = x + 1/2^i \pmod 1$ for every $i \geq 1$. Given a set of points $V \subset [0, 1)$, we define the region $S(v)$ associated with point v as the interval $(\text{pred}(v), v)$, where pred(v) is the closest predecessor of v in V and U is seen as a ring. The following result follows from Ref. [9]:

Theorem 72.1

If every peer is given a random point in $[0, 1)$, then the graph $G_F(V)$ with $|V| = n$ resulting from the continuous-discrete approach has a degree of $O(\log^2 n)$, a diameter of $O(\log n)$, and an expansion of $\Omega(1/\log n)$, with high probability.

Suppose that Condition 72.2 is satisfied for our family of hypercubic functions. Then it is fairly easy to route a message from any point $x \in [0, 1)$ to any point $y \in [0, 1)$ along edges in $G_F(V)$.

Consider the path P in the continuous space that results from using a bit adjustment strategy to get from x to y. That is, given that $x_1x_2x_3\ldots$ is the bit sequence for x and $y_1y_2y_3\ldots$ the bit sequence for y, P is the sequence of points $z_0 = x_1x_2x_3\ldots$, $z_1 = y_1x_2x_3\ldots$, $z_2 = y_1y_2x_3\ldots, \ldots, y_1y_2y_3\ldots = y$. Of course, P may have an infinite length, but simulating P in $G_F(V)$ only requires traversing a finite sequence of edges.

We start with the region $S(v)$ containing $x = z_0$. Then we move along the edge (v, w) in $G_F(V)$ to the region $S(w)$ containing z_1. This edge must exist because we assume that Condition 72.2 is satisfied. Then we move along the edge (w, w') simulating (z_1, z_2), and so on, until we reach the node whose region contains y. Using this strategy, it holds.

Theorem 72.2

Given a random node set $V \subset [0, 1)$ with $|V| = n$, it takes at most $O(\log n)$ hops, with high probability, to route in $G_F(V)$ from any node $v \in V$ to any node $w \in V$.

Next we explain how nodes can join and leave. Suppose that a new node v contacts some node already in the system to join the system. Then v's request is first sent to the node u in V with $u = \mathrm{succ}(v)$, which only takes $O(\log n)$ hops according to Theorem 72.2. u forwards information about all of its incoming and outgoing edges to v, deletes all edges that it does not need any more, and informs the corresponding endpoints about this. Because $S(v) \subseteq S(u)$ for the old $S(u)$, the edges reported to v are a superset of the edges that it needs to establish. v checks which of the edges are relevant for it, informs the other endpoint for each relevant edge, and removes the others.

If a node v wants to leave the network, it simply forwards all of its incoming and outgoing edges to $\mathrm{succ}(v)$. $\mathrm{succ}(v)$ will then merge these edges with its existing edges and notifies the endpoints of these edges about the changes.

Combining Theorems 72.1 and 72.2 we obtain the following theorem:

Theorem 72.3

It takes a routing effort of $O(\log n)$ hops and an update work of $O(\log^2 n)$ messages that can be processed in $O(\log n)$ communication rounds in order to execute a join or leave operation.

72.4.1.2 Maintaining a Dynamic deBruijn Graph

Next, we show how to dynamically maintain a deBruijn graph. An example of a dynamic de Bruijn network (only some shortcut pointers for two nodes are given) is given in Figure 72.2. Let $U = [0, 1)$ and F consist

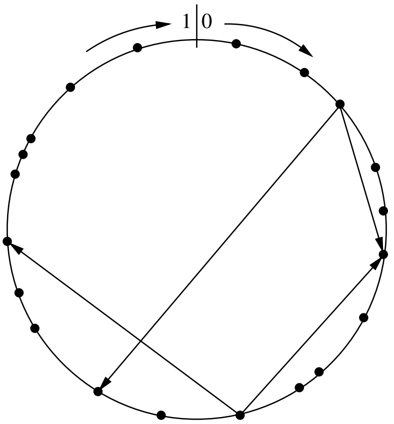


FIGURE 72.2 An example of a dynamic de Bruijn network (only some shortcut pointers for two nodes are given).

of two functions, f_0 and f_1, where $f_i(x) = (i + x)/2$ for each $i \in \{0, 1\}$. Then one can show the following result:

Theorem 72.4

If the peers are mapped to random points in $[0, 1)$, then the graph $G_F(V)$ resulting from the continuous-discrete approach has a degree of $O(\log n)$, diameter of $O(\log n)$, and node expansion of $\Omega(1/\log n)$, with high probability.

Next we show how to route in the de Bruijn network. Suppose that Condition 72.2 is satisfied. Then we use the following trick to route a message from any point $x \in [0, 1)$ to any point $y \in [0, 1)$ along edges in $G_F(V)$.

Let z be a randomly chosen point in $[0, 1)$. Let $x_1x_2x_3\ldots$ be the binary representation of x, $y_1y_2y_3\ldots$ the binary representation of y, and $z_1z_2z_3$ the binary representation of z. Let P be the path along the points $x = x_1x_2x_3\ldots, z_1x_1x_2\ldots, z_2z_1x_1\ldots, \ldots$ and P' the path along the points $y = y_1y_2y_3\ldots, z_1y_1y_2\ldots, z_2z_1y_1\ldots, \ldots$. Then we simulate moving along the points in P by moving along the corresponding edges in $G_F(V)$ until we hit a node $w \in V$ with the property that $S(w)$ contains a point in P and a point in P'. At that point, we follow the points in P' backwards until we arrive at the node $w' \in W$ that contains y in $S(w')$. Using this strategy, it holds.

Theorem 72.5

Given a random node set $V \subset [0, 1)$ with $|V| = n$, it takes at most $O(\log n)$ hops, with high probability, to route in $G_F(V)$ from any point $x \in [0, 1)$ to any point $y \in [0, 1)$.

Joining and leaving the network is done in basically the same way as in the hypercube, giving the following result:

Theorem 72.6

It takes a routing effort of $O(\log n)$ hops and an update work of $O(\log n)$ messages that can be processed in a logarithmic number of communication rounds in order to execute a join or leave operation in the dynamic de Bruijn graph.

72.4.2 Overlay Networks Based on Prefix Connections

The efficiency of routing on an overlay network is quite often measured in terms of the number of hops (neighbor links) followed by a message. While this measure indicates the latency of a message in the overlay network, it fails to convey the complexity of the given operation with respect to the original underlying network. In other words, while in the overlay network all overlay links may have the same cost, this is not true when those overlay links are translated back into paths in the underlying network. Hence, if one is to analyze the routing performance of an overlay network in a way that is more meaningful for real-life scenarios, one needs to take into account the different internode communication costs in the underlying network when charging for the cost of following a path in the overlay network. For example, a hop from a peer in the United States to a peer in Europe costs a lot more (in terms of reliability, speed, cost of deploying and maintaining the link, and so on) than a hop going between two peers in a local area network. In brief, keeping routing local is important: It may make sense to route a message originated in Phoenix for a destination peer in San Francisco through Los Angeles, but not through a peer in Europe.

In this section, we present peer-to-peer overlay network design schemes which take locality into account and which are able to achieve constant stretch factors, while keeping polylogarithmic degree and diameter, and polylogarithmic complexity for join and leave operations. All of the work in this section assumes that the underlying peer-to-peer system is a growth-bounded network, which we define a few paragraphs later.

Peers communicate with one another by means of messages; each message consists of at least one word. We assume that the underlying network supports reliable communication. We define the cost of communication by a function $c : V^2 \rightarrow \Re$. This function c is assumed to reflect the combined effect of the relevant network parameter values, such as latency, throughput, and congestion. In other words, for any two peers u and v in V, $c(u, v)$ is the cost of transmitting a single-word message from u to v. We assume that c is symmetric and satisfies the triangle inequality. The cost of transmitting a message of length l from peer u to peer v is given by $f(l)c(u, v)$, where $f : \mathbf{N} \rightarrow \Re^+$ is any nondecreasing function such that $f(1) = 1$.

A *growth-bounded* network satisfies the following property: Given any u in V and any real r, let $B(u, r)$ denote the set of peers v such that $c(u, v) \leq r$. We refer to $B(u, r)$ as the *ball* of *radius* r around u. We assume that there exists a real constant Δ such that for any peer u in V and any real $r \geq 1$, we have

$$|B(u, 2r)| \leq \Delta |B(u, r)| \tag{72.1}$$

In other words, the number of peers within radius r from u grows polynomially with r. This network model has been validated by both theoreticians and practitioners as to model well-existing internetworking topologies [7,8,10,11].

Plaxton, Rajaraman, and Richa (PRR) in Refs. [5,6] pioneered the work on locality-aware routing schemes in dynamic environments. Their work actually addresses a more general problem—namely the *object location problem*—than that of designing efficient overlay networks with respect to the parameters outlined in Section 72.1. In that early work, Plaxton et al. formalize the problem of object location in a peer-to-peer environment, pinpointing the issue of locality and developing a formal framework under which object location schemes have been rigorously analyzed. In the object location problem, peers seek to find objects in a dynamic and fully distributed environment, where multiple (identical) copies of an object may exist in the network: The main goal is to be able to locate and find a copy of an object within cost that is proportional to the cost of retrieving the closest copy of the object to the requesting peer, while being able to efficiently support these operations in a dynamic peer-to-peer environment where copies of the objects are continuously inserted and removed from the network.

Our overlay network design problem can be viewed as a subset of the object location problem addressed by PRR, where each object is a peer (and hence there exists a single copy of each object in the network). Hence the results in the PRR scheme and other object location schemes to follow, in particular the LAND scheme to be addressed later in this section, directly apply to our overlay network design problem. Both PRR and LAND assume a growth-bounded network model. For these networks, the LAND scheme provides the best currently known overlay design scheme with $1 + \epsilon$ stretch, and polylogarithmic bounds on diameter and degree, for any fixed constant $\epsilon > 0$. Combining the LAND scheme with a technique by Hildrum et al. [12] to find nearest neighbors enable us to also attain polylogarithmic work for JOIN and LEAVE operations. Since the LAND scheme heavily relies on the PRR scheme, we will present the latter in more detail and then highlight the changes introduced by the LAND scheme. We will address both schemes under the light of overlay routing, rather than the object location problem originally addressed by these schemes.

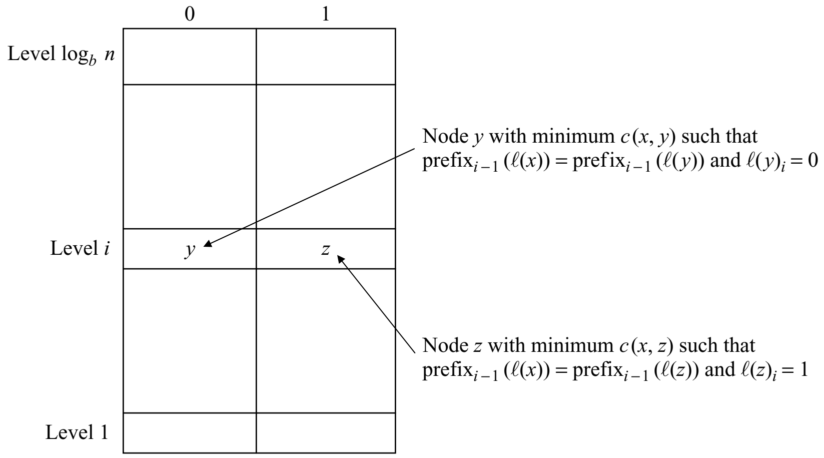
In the PRR and related schemes, each peer p will have two attributes in addition to its exact location in the network (which is unknown to other peers): A virtual location x in U, and a *label* $\ell(x)$ *generated independently and uniformly at random.* We call the virtual location x of p the *peer identifier* x, or simply, the ID x. In the remainder of this section, we will *indistinctly use the* ID x of a peer p *to denote the peer* p *itself.*

72.4.2.1 The PRR Scheme

The original PRR scheme assumes that we have a growth-bounded network with the extra assumption of also having a lower bound on the rate of growth. Later work that evolved from the PRR scheme (e.g., the LAND scheme) showed that this assumption could be dropped by slightly modifying the PRR scheme. The PRR scheme and the vast majority of provably efficient object location schemes rely on a basic yet powerful technique called *prefix routing*, which we outlined in Section 72.4.2. Below, we revisit prefix routing in the context of the PRR scheme.

Theorem 72.7

The PRR scheme, when combined with a technique by Hildrum et al. for finding nearest neighbors, achieves an overlay peer-to-peer network with the following properties: expected constant stretch for ROUTE *operations,* $O(\log n)$ *diameter, and, with high probability,* $O(\log^2 n)$ *degree and work for* JOIN, LEAVE, *and* ROUTE *operations.*



Node y with minimum $c(x, y)$ such that $\text{prefix}_{i-1}(\ell(x)) = \text{prefix}_{i-1}(\ell(y))$ and $\ell(y)_i = 0$

Node z with minimum $c(x, z)$ such that $\text{prefix}_{i-1}(\ell(x)) = \text{prefix}_{i-1}(\ell(z))$ and $\ell(z)_i = 1$

FIGURE 72.3 The neighbor table of peer node x for $b = 2$.

72.4.2.1.1 *Prefix Routing*

The basic idea behind prefix routing (see also Section 72.4.2) is that the path followed in a routing operation will be guided solely by the ID y we are seeking for: Every time a ROUTE(m, y) request is forwarded from a peer u to a peer v in the overlay path, the prefix of $\ell(v)$ that (maximally) matches a prefix of the ID y is strictly larger than that of $\ell(u)$.

We now sketch the PRR prefix routing scheme (for more details, see Ref. [6]). Each peer x in the network is assigned a ($\log_b n$)-digit label[1] $\ell(x) = \ell(x)_1 \ldots \ell(x)_{\log_b n}$, where each $\ell(x)_i \in \{0, \ldots, b-1\}$, uniformly at random, for some large enough constant $b \geq 2$. Recall that each peer also has a unique ($\log_b n$)-digit ID which is independent of this label. We denote the ID of peer x by $x_1 \ldots x_{\log_b n}$, where each $x_i \in \{0, \ldots, b-1\}$.

The random labels are used to construct a *neighbor table* at each peer. For a base b sequence of digits $\gamma = \gamma_1 \ldots \gamma_m$, $\text{prefix}_i(\gamma)$ denotes the first i digits, $\gamma_1 \ldots \gamma_i$, of γ. For each peer x, each integer i between 1 and $\log_b n$, and each digit α between 0 and $b-1$, the neighbor table at peer x stores the ($\log_b n$)-digit ID of the *closest* peer y to x—that is, the peer with minimum $c(x, y)$—such that $\text{prefix}_{i-1}(\ell(x)) = \text{prefix}_{i-1}(\ell(y))$ and $\ell(y)_i = \alpha$. We call y the (i, α)*-neighbor* of peer x. There exists an edge between any pair of neighbor peers in the overlay network. Figure 72.3 illustrates the neighbor table of peer x for $b = 2$.

The degree of a peer in the overlay network is given by the size (number of entries) of its neighbor table. The size of the neighbor table at each peer, as constructed above, is $O(\log n)$. In the final PRR scheme (and other follow-up schemes) the neighbor table at a peer will have size polylogarithmic on n (namely $O(\log^2 n)$ in the PRR scheme), since a set of "auxiliary" neighbors at each peer will need to be maintained in order for the scheme to function efficiently. Each peer x will also need to maintain a set of "pointers" to the location of the subset of peers that were published at x by JOIN operations. In the case of routing, each peer will maintain $O(\log^2 n)$ such pointers with high probability.[2] We describe those pointers in more detail while addressing the JOIN operation in PRR.

The sequence of peers visited in the overlay network during a routing operation for ID y initiated by peer x will consist of the sequence $x = x^0, x^1, \ldots, x^q$, where x^i is the (i, y_i)-neighbor of peer x^{i-1}, for $1 \leq i \leq q$, and x^p is a peer that holds a pointer to y in the network ($p \leq \log_b n$). We call this sequence the

[1] Without loss of generality, assume that n is a power of b.

[2] With probability at least $1 - 1/p(n)$, where $p(n)$ is a polynomial function on n.

neighbor sequence of peer x for (peer ID) y. Below we explain in more detail the ROUTE, JOIN, and LEAVE operations according to PRR.

72.4.2.1.2 *Route, Join, and Leave*

Before we describe a routing operation in the PRR scheme, we need to understand how JOIN and LEAVE operations are processed. Since we are interested in keeping low stretch, the implementation of such operations will be different in the PRR scheme than in the two other overlay network design techniques presented in this chapter. In a ROUTE(m, x) operation, we will, as in the hierarchical decomposition and continuous-discrete approaches, start by routing towards the virtual location x; however, as we route toward this virtual location, as soon as we find some information on the network regarding the actual location of the peer p corresponding to x, we will redirect our route operation to reach p. This way, we will be able to show that the total stretch of a route operation is low (If we were to route all the way to the virtual location x to find information about the actual location of p, the total incurred stretch might be too large.).

There are two main components in a JOIN(p) operation: First, information about the location of peer p joining the peer-to-peer system needs to be published at the network so that subsequent ROUTE operations can indeed locate peer p; second, peer p needs to build its own neighbor table, and other peers in the network may need to update their neighbor tables given the presence of peer x in the network. Similarly, there are two main components in a LEAVE(p) operation: Unpublishing any information about p's location in the network, and removing any entries containing peer p in the neighbor tables of other peers. We will address these two issues separately. For the moment, we will only be concerned with how information about p is published or unpublished in the network, since this is what we need to guarantee the success of a ROUTE operation. We will assume that the respective routing table entries are updated correctly upon the addition or removal of p from the system. Later, we will explain how the neighbor tables can be efficiently updated.

Whenever a peer p with ID x decides to join the peer-to-peer system, we place (*publish*) a pointer leading to the actual location of p in the network, which we call an *x-pointer*, at up to $\log_b n$ peers of the network. For convenience of notation, in the remainder of this section, we will *always use x to indistinctly denote both the ID of peer p and the peer p itself.* Let $x^0 = x, x^1, \ldots, x^{\log_b n-1}$ be the neighbor sequence of peer x for node ID x. We place an x-pointer to x^{i-1} at *each* peer x^i in this neighbor sequence. Thus, whenever we find a peer with an x-pointer (at a peer x^j, $1 \leq j \leq \log_b n$) during a ROUTE(m, x) operation, we can forward the message all the way "down" the reverse neighbor sequence $x^j, \ldots, x^0 = x$ to the actual location of peer x.

The LEAVE(p) operation is the reverse of a JOIN operation: We simply *unpublish* (remove) all the x-pointers from the peers $x^0, \ldots, x^{\log_b n}$.

There is an implicit search tree associated with each peer ID y in prefix routing. Assume for a moment that there is only one peer r matching a prefix of the ID y in the largest number of digits in the network. At the end of this section, we address the case when this assumption does not hold. Let the search tree $T(y)$ for peer y be defined by the network edges in the neighbor sequences for peer ID y for each peer x in the network, where, for each edge of the type (x^{i-1}, x^i) in the neighbor sequence of x for y, we view x^i as the parent of x^{i-1} in the tree. The above implementation of the publish and unpublish operations trivially maintain the following invariant. Let $T_x(y)$ be the subtree rooted at x in $T(y)$.

Invariant 72.1

If peer y belongs to $T_x(y)$, then peer x has a y-pointer.

We now describe how a ROUTE(m, y) operation initiated at peer x proceeds. Let $x^0 = x, x^1, \ldots, x^{\log_b n-1}$ be the neighbor sequence of peer x for y. Starting with $i = 0$, peer x^i first checks whether it has a y-pointer. If it does then x^i will forward the message m using its y-pointer down the neighbor sequence that was used when publishing information about peer y during a JOIN(y) operation. More specifically, let j be the maximum index such that $\text{prefix}_j(\ell(x^i)) = \text{prefix}_j(y)$ (note that $j \geq i$). Then x^i must be equal to y^j, where $y = y^0, y^1, \ldots$ is the neighbor sequence of peer y for the ID y, used during JOIN(y). Thus message m will be forwarded using the y-pointers at $y^j, \ldots, y^0 = y$ (if y^j has a y-pointer, then so does y^k for all $1 \leq k \leq j$) all the way down to peer y. If x^i does not have a y-pointer, it will simply forward the message m to x^{i+1}.

Given Invariant 72.1, peer x will locate peer y in the network if peer y is indeed part of the peer-to-peer system. The cost of routing to peer y can be bounded by the following fact:

Fact 72.7

A message from peer x to peer y will be routed through a path with cost $O(\sum_{k=1}^{j}[c(x^{k-1}, x^k)+c(y^{k-1}, y^k)])$.

A deficiency of this scheme, as described, is that there is a chance that we may fail to locate a pointer to y at x^1 through $x^{\log_b n}$. In this case, we must have more than one "root peer" in $T(y)$ (a root peer is a peer such that the length of its maximal prefix matching the ID of y is maximum among all peers in the network), and hence there is no guarantee that there exists a peer r which will have a global view of the network with respect to the ID of peer y. Fortunately, this deficiency may be easily rectified by a slight modification of the algorithm, as shown in Ref. [6].

The main challenge in the analysis of the PRR scheme is to show that the summation in Fact 72.7 is indeed $O(c(x, y))$. The probability that the k-digit prefix of the label of an arbitrary peer matches a particular k-digit string $\gamma = \text{prefix}_k(y)$, for some peer y, is b^{-k}. Consider a ball B around peer x^{k-1} containing exactly b^k peers. Note that there is a constant probability (approximately $1/e$) that no peer in B matches γ. Thus the radius of B is a lower bound (up to a constant factor) on the expected distance from x^{k-1} to its (k, y_k)-neighbor. Is this radius also an upper bound? Not for an arbitrary metric, since (for example) the diameter of the smallest ball around a peer z containing $b^k + 1$ peers can be arbitrarily larger than the diameter of the smallest ball around z containing b^k peers. However, it can be shown that the radius of B provides a tight bound on the expected distance from x^{k-1} to its (k, y_k)-neighbor. Furthermore, Eq. (72.1) implies that the expected distance from x^{k-1} to its (k, y_k)-neighbor is geometrically increasing in k. The latter observation is crucial since it implies that the expected total distance from x^{k-1} to its (k, y_k)-neighbor, summed over all k such that $1 \leq k \leq j$, is dominated by (i.e., within a constant factor of) the expected communication cost from x^{j-1} to its (j, y_j)-neighbor x^j.

We still need to show that, $c(x, y) = O(E[c(x^{j-1}, x^j)])$, and hence that the routing stretch factor in the PRR scheme is constant in expectation. This proof is technically rather involved and we refer the reader to Ref. [6]. In the next section, we will show how the PRR scheme can be elegantly modified to yield deterministic constant stretch. More specifically, the LAND scheme achieves deterministic stretch $1 + \epsilon$, for any fixed $\epsilon > 0$.

72.4.2.1.3 *Updating the Neighbor Tables*

To be able to have JOIN and LEAVE operations with low work complexity, while still enforcing low stretch ROUTE operations and polylogarithmic degree, one needs to devise an efficient way for updating the neighbor tables upon the arrival or departure of a peer from the system. The PRR scheme alone does not provide such means. Luckily, we can combine the work by Hildrum et al. [12], which provides an efficient way for finding nearest neighbors in a dynamic and fully distributed environment, with the PRR scheme to be able to efficiently handle the insertion or removal of a peer from the system. Namely, the work by Hildrum et al. presents an algorithm which can build the neighbor table of a peer p joining the system and update the other peers' neighbor tables to account for the new peer in the system with total work $O(\log^2 n)$.

72.4.2.2 The LAND Scheme

The LAND scheme proposed by Abraham et al. in Ref. [10] is the first, and currently best-known, peer-to-peer overlay network design scheme to achieve constant deterministic stretch for routing, while maintaining polylogarithmic diameter, degree, and work (for ROUTE, JOIN, and LEAVE) for growth-bounded metrics. Note that LAND does not require a lower bound on the growth as PRR does. Like the PRR scheme, the LAND scheme was also designed for the more general problem of object location in peer-to-peer systems.

Namely, the main results of the LAND scheme are summarized in the following theorem:

Theorem 72.8

The LAND scheme, when combined with a technique by Hildrum et al. for finding nearest neighbors, achieves an overlay peer-to-peer network with the following properties: deterministic $(1+\epsilon)$ *stretch for* ROUTE *operations, for any fixed* $\epsilon > 0$, $O(\log n)$ *diameter, and expected* $O(\log n)$ *degree and work for* JOIN, LEAVE, *and* ROUTE *operations.*

The LAND scheme is a variant of the PRR scheme. The implementation of ROUTE, JOIN, and LEAVE operations in this later scheme are basically the same as in PRR. The basic difference between the two schemes is on how the peer labels are assigned to the peers during the neighbor table construction phase. A peer may hold more than one label, some of which may not have been assigned in a fully random and independent way.

In a nutshell, the basic idea behind the LAND scheme is that instead of letting the distance between a peer x and its (i, α)-neighbor be arbitrarily large, it will enforce that this distance be always at most some constant β times b^i by letting peer x emulate a virtual peer with label γ such that $\text{prefix}_i(\gamma) = x_1 \ldots x_{i-1}\alpha$ if no peer has $x_1 \ldots x_{i-1}\alpha$ as a prefix of its label in a ball centered at x with $O(b^i)$ peers in it. Another difference in the LAND scheme which is crucial to guarantee a deterministic bound on stretch is that the set of "auxiliary" neighbors it maintains are only used during join/leave operations, rather than during routing operations such as in the PRR scheme.

The analysis of the LAND scheme is elegant, consisting of short and intuitive proofs. Thus, this is also a main contribution of this scheme, given that the analysis of the PRR scheme is rather lengthy and involved.

72.5 Other Related Work

There is a wealth of literature on peer-to-peer systems, and papers on this subject can be found in every major computer science conference. Peer-to-peer overlay networks can be classified into three categories: social networks, random networks, and structured networks.

Examples of social networks are Gnutella and KaZaA. Their basic idea of interconnecting peers is that connections follow the principle of highest benefit: a peer preferably connects to peers with similar interests by maintaining direct links to those peers that can successfully answer queries.

An example for random networks is JXTA, a Java library developed by SUN to facilitate the development of peer-to-peer systems. The basic idea behind the JXTA core is to maintain a random-looking network between the peers. In this way, peers are very likely to stay in a single connected component because random graphs are known to be robust against even massive failures or departures of nodes. Recent theory work on random peer-to-peer networks can be found in Refs. [13,14].

Most of the scientific work on peer-to-peer networks has focused on structured overlay networks, that is, networks with a regular structure that makes it easy to route in them with low overhead. The vast majority of these networks is based on the concept of virtual space. The most prominent among these are Chord [4], CAN [1], Pastry [7], and Tapestry [8]. The virtual space approach has the problem that it requires node labels to be evenly distributed in the space to obtain a scalable overlay network. Alternative approaches that yield scalable overlay networks for arbitrary distinct node labels are skip graphs [15], skip nets [16], and the hyperring [17].

As we have already seen above, structured overlay networks are also known that can take locality into account. All of this work is based on the results in Refs. [5,6]. The first peer-to-peer systems were Tapestry [8] and Pastry [7]. Follow-up schemes addressed some of the shortcomings of the PRR scheme. In particular, the LAND scheme [10] improves the results in PRR as seen in the previous section; algorithms for efficiently handling peer arrival and departures are presented in Refs. [12,18]; a simplified scheme with provable bounds for ring networks is given in Ref. [19]; a fault-tolerant extension is given in Ref. [20]; a scheme that addresses general networks, at the expense of an $O(\log n)$ stretch bound, is given in Ref. [21]; a scheme that considers object location under more realistic networks is given in Ref. [22]; and a first attempt at designing overlay networks in peer-to-peer systems consisting of mobile peers is presented in Ref. [23] (no formal bounds are proven in Ref. [23] for any relevant network distribution though).

Finally, overlay networks have not only been designed for wired networks but also for wireless networks. See, for example, Ref. [24] and references therein for newest results in this area.

Acknowledgment

Supported by NSF CAREER grant CCR-9985284.

References

[1] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S., A scalable content-addressable network, *Proc. of SIGCOMM*, 2001.

[2] Wang, X., Zhang, Y., Li, X., and Loguinov, D., On zone-balancing of peer-to-peer networks: analysis of random node join, *Proc. of SIGMETRICS*, 2004.

[3] Naor, M. and Wieder, U., Novel architectures for P2P applications: the continuous-discrete approach, *Proc. of SPAA,* 2003.

[4] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H., Chord: a scalable peer-to-peer lookup service for Internet applications, *Proc. of SIGCOMM*, 2001.

[5] Plaxton, C. G., Rajaraman, R., and Richa, A. W., Accessing nearby copies of replicated objects in a distributed environment, *Proc. of SPAA*, 1997, p. 311.

[6] Plaxton, C., Rajaraman, R., and Richa, A., Accessing nearby copies of replicated objects in a distributed environment, *Theor. Comput. Syst.*, 32, 241, 1999.

[7] Druschel, P. and Rowstron, A., Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, *Proc. Int. Conf. on Distributed Systems Platforms*, 2001.

[8] Zhao, B. Y., Kubiatowicz, J., and Joseph, A., Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing, TR, University of California at Berkeley, CS Department, 2001.

[9] Awerbuch, B. and Scheideler, C., Chord++: Low-congestion routing in chord, Unpublished manuscript, Johns Hopkins University, June 2003 (see also http://www.in.tum.de/personen/scheideler/index.html.en).

[10] Abraham, I., Malkhi, D., and Dobzinski, O., LAND: stretch $(1+\epsilon)$ locality aware networks for DHTs, *Proc. of SODA*, 2004.

[11] Karger, D. R. and Ruhl, M., Finding nearest neighbors in growth-restricted metrics, *Proc. of STOC*, 2002, p. 741.

[12] Hildrum, K., Kubiatowicz, J., Rao, S., and Zhao, B. Y., Distributed object location in a dynamic network, *Proc. of SPAA*, 2002, p. 41.

[13] Mahlmann, P. and Schindelhauer, C., Peer-to-peer networks based on random transformations of connected regular undirected graphs, *Proc. of SPAA,* 2005, p. 155.

[14] Cooper, C., Dyer, M., and Greenhill, C., Sampling regular graphs and a peer-to-peer network, *Proc. of SODA*, 2005.

[15] Aspnes, J. and Shah, G., Skip graphs, *Proc. of SODA*, 2003, p. 384.

[16] Harvey, N. J., Jones, M. B., Saroiu, S., Theimer, M., and Wolman, A., Skipnet: a scalable overlay network with practical locality properties, *Proc. Symp. on Internet Technologies and Systems*, 2003.

[17] Awerbuch, B. and Scheideler, C., The Hyperring: a low-congestion deterministic data structure for distributed environments, *Proc. of SODA*, 2004.

[18] Hildrum, K., Kubiatowicz, J., Ma, S., and Rao, S., A note on the nearest neighbor in growth-restricted metrics, *Proc. of SODA*, 2004, p. 560.

[19] Li, X. and Plaxton, C. G., On name resolution in peer-to-peer networks, *Proc. Worskhop on Principles of Mobile Commerce*, 2002, p. 82.

[20] Hildrum, K. and Kubiatowicz, J., Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks, *Proc. of DISC*, 2003, p. 321.

[21] Rajaraman, R., Richa, A. W., Vöcking, B., and Vuppuluri, G., A data tracking scheme for general networks, *Proc. of SPAA,* 2001, p. 247.

[22] Hildrum, K., Krauthgamer, R., and Kubiatowicz, J., Object location in realistic networks, *Proc. of SPAA*, 2004, p. 25.

[23] Abraham, I., Dolev, D., and Malkhi, D., LLS: a locality aware location service for mobile ad hoc networks, *Proc. of DIALM-POMC*, 2004.

[24] Scheideler, C., Overlay networks for wireless ad hoc networks, *Proc. IMA Workshop on Wireless Communications,* 2005 (see also http://www.in.tum.de/personen/scheideler/index.html.en).

73

Scheduling Data Broadcasts on Wireless Channels: Exact Solutions and Heuristics*

Alan A. Bertossi
University of Bologna

M. Cristina Pinotti
University of Perugia

Romeo Rizzi
University of Udine

73.1 Introduction

We discuss the broadcasting problem of N data items over K wireless channels, under the assumptions of skewed data allocation to channels and flat data scheduling per channel. Both the uniform and nonuniform length cases are surveyed showing their exact and heuristic solutions, respectively. Two of the heuristic methods are greedy and the third one is based on a dynamic programming procedure developed to solve a simplified version of the problem. An experimental evaluation of our heuristics is presented and the quality of the solutions generated is compared against a lower bound, which is derived by relaxing the problem and then solving it optimally via a dynamic programming procedure developed in earlier sections.

In wireless asymmetric communication, broadcasting is an efficient way of simultaneously disseminating data to a large number of clients. Consider data services on cellular networks, such as stock quotes, weather information, traffic news, where data are continuously broadcast to clients that may desire them at any instant of time. In this scenario, a server at the base-station repeatedly transmits data items from a given set over a wireless channel, while clients passively listen to the shared channel waiting for their desired item. The server follows a broadcast schedule for deciding which item of the set has to be transmitted at any time instant. An efficient broadcast schedule minimizes the client expected delay, that is, the average amount of time spent by a client before receiving the item needed. The client expected delay increases with the size of the set of the data items to be transmitted by the server. Indeed, the client has to wait for many unwanted data before receiving its own data. The efficiency can be improved by augmenting the

*Portions of this chapter are reprinted, with permission, from Ref. [1] *IEEECS Log Number TC-0238-0704*, © 2005, IEEE.

server bandwidth, for example, allowing the server to transmit over multiple disjoint physical channels and therefore defining a shorter schedule for each single channel. In a multichannel environment, in addition to a broadcast schedule for each single channel, an allocation strategy has to be pursued so as to assign data items to channels. Moreover, each client can access either only a single channel or any available channel at a time. In the former case, if the client can access only one prefixed channel and can potentially retrieve any available data, then all data items must be replicated over all channels. Otherwise, data can be partitioned among the channels, thus assigning each item to only one channel. In this latter case, the efficiency can be improved by adding an index that informs the client at which time and on which channel the desired item will be transmitted. In this way, the mobile client can save battery energy and reduce the tuning time because, after reading the index info, it can sleep and wake up on the proper channel just before the transmission of the desired item.

Several variants for the problem of data allocation and broadcast scheduling have been proposed in the literature, which depend on the perspectives faced by the research communities [2–12].

Specifically, the networking community faces a version of the problem, known as the *Broadcast Problem*, whose goal is to find an infinite schedule on a single channel [4,6,7,10]. Such a problem was first introduced in the teletext systems [2,3]. Although it is widely studied (e.g., it can be modeled as a special case of the Maintenance Scheduling Problem and the MultiItem Replenishment Problem [4,6]), its tractability is still under consideration. Therefore, the emphasis is on finding near optimal schedules for a single channel. Almost all the proposed solutions follow the *square root rule (SRR)* [3]. The aim of such a rule is to produce a broadcast schedule where each data item appears with equally spaced replicas, whose frequency is proportional to the square root of its popularity and inversely proportional to the square root of its length. The multichannel schedule is obtained by distributing in a round robin fashion the schedule for a single channel [10]. Since each item appears in multiple replicas which, in practice, are not equally spaced, these solutions make indexing techniques ineffective. Briefly, the main results known in the literature for the Broadcast Problem can be summarized as follows. For *uniform* lengths, namely all items of the same length, it is still unknown whether the problem can be solved in polynomial time or not. For a constant number of channels, the best algorithm proposed so far is the polynomial-time approximation scheme (PTAS) devised in Ref. [7]. In contrast, for *nonuniform* lengths, the problem has been shown to be strong NP-hard even for a single channel, a 3-approximation algorithm was devised for one channel, and a heuristic has been proposed for multiple channels [6].

On the other hand, the database community seeks for a periodic broadcast scheduling which should be easily indexed [5]. For the single channel, the obvious schedule that admits index is the *flat* one. It consists in selecting an order among the data items, and then transmitting them once at a time, in a round-robin fashion [13], producing an infinite periodic schedule. In a flat schedule indexing is trivial, since each item will appear once, and exactly at the same relative time, within each period. Although indexing allows the client to sleep and save battery energy, the client expected delay is half of the schedule period and can become infeasible for a large period. To decrease the client expected delay, still preserving indexing, flat schedules on multiple channels can be adopted [8,9,12]. However, in such a case the allocation of data to channels becomes critical. For example, allocating items in a balanced way simply scales the expected delay by a factor equal to the number of channels. To overcome this drawback, *skewed* allocations have been proposed where items are partitioned according to their popularity so that the most requested items appear in a channel with shorter period [8,12]. Hence, the resulting problem is slightly different from the Broadcast Problem since, to minimize the client expected delay, it assumes skewed allocation and flat scheduling. This variant of the problem is easier than the Broadcast Problem. Indeed, as proved in Ref. [12], an optimal solution for uniform lengths can be found in polynomial time. In contrast, the problem becomes computationally intractable for nonuniform lengths [1]. For this latter case, several heuristics have been developed in Refs. [12,14], which have been tested on some benchmarks whose item popularity follow Zipf distributions. Such distributions are used to characterize the popularity of one element among a set of similar data, like a web page in a web site [15].

The present chapter reviews the work of Refs. [1,12,14] on the broadcasting problem of N data items over K wireless channels, under the assumptions of skewed data allocation to channels and flat data scheduling per channel. Both the uniform and nonuniform length cases are surveyed showing their exact and heuristic solutions, respectively. Two of the heuristic methods are greedy and the third one is based on a dynamic programming procedure used to solve a simplified version of the problem. An experimental evaluation of our heuristics is presented and the quality of the solutions generated compared against a lower bound which is derived by relaxing the problem and then solving it optimally via a dynamic programming procedure developed in earlier sections.

For the case of data items with uniform lengths, three exact polynomial time algorithms are presented, all based on dynamic programming. The first algorithm, called *DP* and originally proposed in Ref. [12], takes $O(N^2 K)$ time, while the second algorithm, called *Dichotomic* and proposed later in Ref. [1], is faster as it runs in $O(NK \log N)$ time. The third algorithm, presented in Ref. [14], is designed for the specific case of $K = 2$. Although it requires $O(N \log N)$ time, and hence it is asymptotically not faster than Dichotomic, it exploits a specific characterization of the optimal solution when there are only two channels.

For the case of data items with nonuniform lengths, the problem is NP-hard when $K = 2$, and *strong* NP-hard for arbitrary K. In this latter case, the *Optimal* algorithm presented in Ref. [1] is reviewed. It requires $O(KN^{2z})$ time, where z is the maximum data length, and reduces to the DP algorithm when $z = 1$. Since algorithm Optimal can solve only small instances in a reasonable time, three heuristics are described, all having an $O(N(K + \log N))$ time complexity.

The first heuristic, called *Greedy*, has been proposed in Ref. [12]. For any fixed N, Greedy starts with all data items assigned to one channel, and then proceeds by splitting the items of one channel between two channels, thus adding a new channel, until K channels are reached. The other two heuristics, both presented in Ref. [14], pretend that the characterization of the optimal solution of the problem for $K = 2$ and uniform lengths holds also for the general case of arbitrary K and nonuniform lengths. One heuristic is called *Greedy+* since it combines such solution characterization with the Greedy approach, while the second heuristic is called *Dlinear* and combines the same characterization with the dynamic programming relation proposed in Ref. [12].

All the three heuristics are then tested on benchmarks whose item popularity are characterized by Zipf distributions. The experimental tests reveal that Dlinear finds optimal solutions almost always, requiring reasonable running times. Although Greedy remains the fastest heuristic, it gives the worst suboptimal solutions. Both the running times and the quality of the solutions of Greedy+ are intermediate between those of Dlinear and Greedy. However, Greedy and Greedy+ have the feature to scale well with respect to the parameter changes.

The rest of this chapter is so organized. Section 73.2 gives notations, definitions, and the problem statement. Section 73.3 illustrates the DP and Dichotomic algorithms for items of uniform lengths, as well as the solution characterization for the particular case of two channels. In contrast, Section 73.4 studies the nonuniform length case. It first recalls the strong NP-hardness for an arbitrary number of channels, and then presents the exponential time Optimal algorithm. Then, Section 73.5 gives the Greedy, Greedy+, and Dlinear heuristics. Section 73.6 reports the experimental tests on some benchmarks, whose item popularity follow Zipf distributions. Finally, we discuss our conclusions in Section 73.7.

73.2 Problem Formulation

Consider a set of K identical channels, and a set $D = \{d_1, d_2, \ldots, d_N\}$ of N data items. Each item d_i is characterized by a *probability* p_i and a *length* z_i, for $1 \leq i \leq N$. The probability p_i represents the popularity of item d_i, namely its probability to be requested by the clients, and it does not vary along the time. Clearly, $\sum_{i=1}^{N} p_i = 1$. The length z_i is an integer number, counting how many time units are required to transmit item d_i on any channel. When all data lengths are the same, that is, $z_i = z$ for $1 \leq i \leq N$, the lengths are called *uniform* and are assumed to be unit, that is, $z = 1$. When the data lengths are not the same, the lengths are said *nonuniform.*

The items have to be partitioned into K groups $G_1, \ldots, G_K$. Group G_j collects the data items assigned to channel j, for $1 \leq j \leq K$. The cardinality of G_j is denoted by N_j, the sum of its item lengths is denoted by Z_j, that is, $Z_j = \sum_{d_i \in G_j} z_i$, and the sum of its probabilities is denoted by P_j, that is, $P_j = \sum_{d_i \in G_j} p_i$. Note that since the items in G_j are cyclically broadcast according to a flat schedule, Z_j is the schedule period on channel j. Clearly, in the uniform case $Z_j = N_j$, for $1 \leq j \leq K$. If item d_i is assigned to channel j, and assuming that clients can start to listen at any instant of time with the same probability, the *client expected delay* for receiving item d_i is half of the period, namely $\frac{Z_j}{2}$. Assuming that indexing allows clients to know in advance the content of the channels [12], the *average expected delay* (AED) over all channels is

$$\text{AED} = \frac{1}{2} \sum_{j=1}^{K} Z_j P_j \tag{73.1}$$

Given K channels, a set D of N items, where each data item d_i comes along with its probability p_i and its integer length z_i, the *K-Nonuniform Allocation Problem* consists in partitioning D into K groups $G_1, \ldots, G_K$, so as to minimize the objective function AED given in Eq. 73.1. In the special case of equal lengths, the above problem is called *K-Uniform Allocation Problem* and the corresponding objective function is derived replacing Z_j with N_j in Eq. (73.1).

As an example, consider a set of $N = 6$ items with uniform lengths and $K = 3$ channels. Let the demand probabilities be $p_1 = 0.37$, $p_2 = 0.25$, $p_3 = 0.18$, $p_4 = 0.11$, $p_5 = 0.05$, and $p_6 = 0.04$. The optimal solution assigns item d_1 to the first channel, items d_2 and d_3 to the second channel, and the remaining items to the third channel. The corresponding AED is $\frac{1}{2}(0.37 + 2(0.25 + 0.18) + 3(0.11 + 0.05 + 0.04)) = 0.915$.

Consider the sequence $d_1, \ldots, d_N$ of items ordered by their indices, and assume that each channel contains items with consecutive indices. Then, the cost of assigning to a single channel consecutive items within the sequence, say from d_i to d_j, is $C_{i,j} = \frac{1}{2}(\sum_{h=i}^{j} p_h)(\sum_{h=i}^{j} z_h)$. Letting $P_{i,j} = \sum_{h=i}^{j} p_h$ and $Z_{i,j} = \sum_{h=i}^{j} z_h$, one notes that all the $P_{1,n}$ and $Z_{1,n}$, for $1 \leq n \leq N$, can be computed in $O(N)$ time by two prefix sum computations. Hence, a single $C_{i,j}$ can be computed on the fly in constant time as $C_{i,j} = \frac{1}{2}(P_{1,j} - P_{1,i-1})(Z_{1,j} - Z_{1,i-1})$. From now on, to simplify the presentation, $C_{i,j}$ is defined to be 0 whenever $i > j$. Note that, for uniform lengths, the formula of $C_{i,j}$ simplifies as $C_{i,j} = \frac{1}{2}(j - i + 1) \sum_{h=i}^{j} p_h$.

Moreover, a *segmentation* is a partition of the ordered sequence $d_1, \ldots, d_N$ into $G_1, \ldots, G_K$, such that if $d_i \in G_k$ and $d_j \in G_k$ then $d_h \in G_k$ whenever $i \leq h \leq j$. A segmentation

$$\underbrace{d_1, \ldots, d_{B_1}}_{G_1}, \underbrace{d_{B_1+1}, \ldots, d_{B_2}}_{G_2}, \ldots, \underbrace{d_{B_{K-1}+1}, \ldots, d_N}_{G_K}$$

is compactly denoted by the $(K-1)$-tuple

$$(B_1, B_2, \ldots, B_{K-1})$$

of its *right borders*, where border B_k is the index of the last item that belongs to group G_k. Note that it is not necessary to specify B_K, the index of the last item of the last group, because its value will be N for any solution. From now on, B_{K-1} will be referred to as the *final border* of the solution. The cardinality of G_k, that is, the number N_k of items in the group, is $N_k = B_k - B_{k-1}$, where $B_0 = 0$ and $B_K = N$ are assumed.

73.3 Uniform Lengths

This section is devoted to examine the dynamic programming algorithms proposed in [1,12,14] for the K-Uniform Allocation Problem. The following results show that the optimal solutions for the K-Uniform Allocation Problem can be sought within the class of segmentations.

Lemma 73.1 (Yee et al. [12])

Let G_h and G_j be two groups in an optimal solution for the K-Uniform Allocation Problem. Let d_i and d_k be items with $d_i \in G_h$ and $d_k \in G_j$. If $N_h < N_j$, then $p_i \geq p_k$. Similarly, if $p_i > p_k$, then $N_h \leq N_j$.

In other words, the most popular items are allocated to less loaded channels so that they appear more frequently. As a consequence, if the items are sorted by nonincreasing probabilities, then the group sizes are nondecreasing.

Corollary 73.1 (**Yee et al.** [12])

Let $d_1, d_2, \ldots, d_N$ be N uniform length items with $p_i \geq p_k$ whenever $i < k$. Then, there exists an optimal solution for partitioning them into K groups $G_1, \ldots, G_K$, where each group is made of consecutive elements.

Thus, assuming the items sorted by nonincreasing probabilities, any sought solution S will be a segmentation. Moreover, the sought segmentations $S = (B_1, B_2, \ldots, B_{K-1})$ for the uniform case can be restricted to those verifying $N_1 \leq N_2 \leq \cdots \leq N_K$, which will be called *feasible* segmentations.

73.3.1 The DP Algorithm

To describe the DP algorithm [12], let $OPT_{n,k}$ denote an optimal solution for grouping items $d_1, \ldots d_n$ into k groups and let $opt_{n,k}$ be its corresponding cost, for any $n \leq N$ and $k \leq K$. Since $d_1, d_2, \ldots, d_N$ are sorted by nonincreasing probabilities, one has

$$opt_{n,k} = \begin{cases} C_{1,n} & \text{if } k = 1 \\ \min_{1 \leq \ell \leq n-1}\{opt_{\ell,k-1} + C_{\ell+1,n}\} & \text{if } k > 1 \end{cases} \tag{73.2}$$

The DP algorithm is a dynamic programming implementation of recurrence (73.2). Indeed, to find $OPT_{n,k}$, consider the $K \times N$ matrix M with $M_{k,n} = opt_{n,k}$. The entries of M are computed row by row applying recurrence (73.2). Clearly, $M_{K,N}$ contains the cost of an optimal solution for the K-Uniform Allocation Problem. To actually construct an optimal partition, a second matrix F is employed to keep track of the final borders of segmentations corresponding to entries of M. In recurrence (73.2), the value of ℓ, which minimizes the right-hand side, is the *final border* for the solution $OPT_{n,k}$ and is stored in $F_{k,n}$. Hence, the optimal segmentation is given by $OPT_{N,K} = (B_1, B_2, \ldots, B_{K-1})$ where, starting from $B_K = N$, the value of B_k is equal to $F_{k+1,B_{k+1}}$, for $k = K-1, \ldots, 1$.

To evaluate the time complexity of the DP algorithm, observe that $O(n)$ comparisons are required to fill the entry $M_{k,n}$, which implies that $O(N^2)$ comparisons are required to fill a row. Since there are K rows, the complexity of the DP algorithm is $O(N^2K)$.

73.3.2 The Dichotomic Algorithm

To improve on the time complexity of the DP algorithm for the K-Uniform Allocation Problem, the properties of optimal solutions have to be further exploited.

Definition 73.1

Let $d_1, d_2, \ldots, d_N$ be uniform length items sorted by nonincreasing probabilities. An optimal solution $OPT_{N,K} = (B_1, B_2, \ldots, B_{K-1})$ is called left-most optimal *and denoted by $LMO_{N,K}$ if, for any other optimal solution $(B'_1, B'_2, \ldots, B'_{K-1})$, it holds $B_{K-1} \leq B'_{K-1}$.*

Clearly, since the problem always admits an optimal solution, there is always a leftmost optimal solution. Although the leftmost optimal solutions do not need to be unique, it is easy to check that there exists a unique $(B_1, B_2, \ldots, B_{K-1})$ such that $(B_1, B_2, \ldots, B_i)$ is a left-most optimal solution for partitioning into $i+1$ groups the items $d_1, d_2, \ldots, d_{B_{i+1}}$, for every $i < K$.

Definition 73.2

A leftmost optimal solution $(B_1, B_2, \ldots, B_{K-1})$ for the K-Uniform Allocation Problem is called strict leftmost optimal solution, *and denoted by $SLMO_{N,K}$, if $(B_1, B_2, \ldots, B_i)$ is a $LMO_{B_{i+1},i+1}$, for every $i < K$.*

The Dichotomic algorithm computes a leftmost optimal solution for every $i < K$, and thus it finds the unique strict leftmost optimal solution.

Lemma 73.2 (Ardizzoni et al. [1])

Let $d_1, d_2, \ldots, d_N$ be uniform length items sorted by nonincreasing probabilities. Let $LMO_{N-1,K} = (B_1, B_2, \ldots, B_{K-1})$ and $OPT_{N,K} = (B'_1, B'_2, \ldots, B'_{K-1})$. Then, $B'_{K-1} \geq B_{K-1}$.

In other words, Lemma 73.2 implies that, given the items sorted by nonincreasing probabilities, if one builds an optimal solution for N items from an optimal solution for $N - 1$ items, then the final border B_{K-1} can only move to the right. Such a property can be easily generalized as follows to problems of increasing sizes. From now on, let B^c_h denote the h-th border of $LMO_{c,k}$, with $k > h \geq 1$.

Corollary 73.2 (Ardizzoni et al. [1])

Let $d_1, d_2, \ldots, d_N$ be uniform length items sorted by nonincreasing probabilities, and let $l < j < r \leq N$. Then, $B^l_{K-1} \leq B^j_{K-1} \leq B^r_{K-1}$.

Corollary 73.2 plays a fundamental role in speeding up the DP algorithm. Indeed, assume that $LMO_{n,k-1}$ has been found for every $1 \leq n \leq N$. If the $LMO_{l,k}$ and $LMO_{r,k}$ solutions are also known for some $1 \leq l \leq r \leq N$, then one knows that B^j_{k-1} is between B^l_{k-1} and B^r_{k-1}, for any $l \leq j \leq r$. Thus, recurrence (73.2) can be rewritten as

$$opt_{j,k} = \min_{B^l_{k-1} \leq \ell \leq B^r_{k-1}} \{opt_{\ell,k-1} + C_{\ell+1,j}\} \tag{73.3}$$

As the name suggests, the $O(KN \log N)$ time Dichotomic algorithm is derived by choosing $j = \lceil \frac{l+r}{2} \rceil$ in recurrence (73.3), thus obtaining:

$$opt_{\lceil \frac{l+r}{2} \rceil,k} = \min_{B^l_{k-1} \leq \ell \leq B^r_{k-1}} \{opt_{\ell,k-1} + C_{\ell+1,\lceil \frac{l+r}{2} \rceil}\} \tag{73.4}$$

where B^l_{k-1} and B^r_{k-1} are, respectively, the final borders of $LMO_{l,k}$ and $LMO_{r,k}$. Such recurrence is iteratively solved within three nested loops which vary, respectively, in the ranges $1 \leq k \leq K$, $1 \leq t \leq \lceil \log N \rceil$, and $1 \leq i \leq 2^{t-1}$, and where the indices l, r, and j are set as follows: $l = \lceil \frac{i-1}{2^{t-1}}(N+1) \rceil$, $r = \lceil \frac{i}{2^{t-1}}(N+1) \rceil$, and $j = \lceil \frac{l+r}{2} \rceil = \lceil \frac{2i-1}{2^t}(N+1) \rceil$.

In details, the Dichotomic algorithm is shown in Figure 73.1. It uses the two matrices M and F, whose entries are again filled up row by row (Loop 1). A generic row k is filled in stages (Loop 2). Each stage corresponds to a particular value of the variable t (Loop 3). The variable j corresponds to the index of the entry, which is currently being filled in stage t. The variables l (left) and r (right) correspond to the indices of the entries nearest to j which have been already filled, with $l < j < r$.

```
Input:        $N$ items sorted by nonincreasing probabilities, and $K$ groups;
Initialize:       for $i$ from 1 to $N$ do
                      for $k$ from 1 to $K$ do
                          if $k = 1$ then $M_{k,i} \leftarrow C_{k,i}$ else $M_{k,i} \leftarrow \infty$;
Loop 1:           for $k$ from 2 to $K$ do
                      $F_{k,0} \leftarrow F_{k,1} \leftarrow 1$; $F_{k,N+1} \leftarrow N$;
Loop 2:               for $t$ from 1 to $\lceil \log N \rceil$ do
Loop 3:                   for $i$ from 1 to $2^{t-1}$ do
                              $j \leftarrow \lceil \frac{2i-1}{2^t}(N+1) \rceil$; $l \leftarrow \lceil \frac{i-1}{2^{t-1}}(N+1) \rceil$; $r \leftarrow \lceil \frac{i}{2^{t-1}}(N+1) \rceil$;
                              if $M_{k,j} = \infty$ then
Loop 4:                           for $\ell$ from $F_{k,l}$ to $F_{k,r}$ do
                                      if $M_{k-1,\ell} + C_{\ell+1,j} < M_{k,j}$ then
                                          $M_{k,j} \leftarrow M_{k-1,\ell} + C_{\ell+1,j}$;
                                          $F_{k,j} \leftarrow \ell$;
```

FIGURE 73.1 The Dichotomic algorithm for the K-Uniform Allocation Problem. (From Ardizzoni, E. et al., *IEEE Trans. Comput.,* 54(5), 558, 2005. *IEEECS Log Number TC-0238-0704,* Copyright 2005, IEEE with permission.)

If no entry before j has been already filled, then $l = 1$, and therefore the final border $F_{k,1}$ is initialized to 1. If no entry after j has been filled, then $r = N$, and thus the final border $F_{k,N+1}$ is initialized to N. To compute the entry j, the variable ℓ takes all values between $F_{k,l}$ and $F_{k,r}$. The index ℓ which minimizes the recurrence in Loop 4 is assigned to $F_{k,j}$, while the corresponding minimum value is assigned to $M_{k,j}$.

To show the correctness, consider how a generic row k is filled up. In the first stage (i.e., $t = 1$), the entry $M_{k,\lceil \frac{N+1}{2} \rceil}$ is filled and ℓ ranges over all values $1, \dots, N$. By Corollary 73.2, observe that to fill an entry $M_{k,l}$, where $l < \lceil \frac{N+1}{2} \rceil$, one needs to consider only the entries $M_{k-1,\ell}$, where $\ell \leq F_{k,\lceil \frac{N+1}{2} \rceil}$. Similarly, to fill an entry $M_{k,l}$, where $l > \lceil \frac{N+1}{2} \rceil$, one needs to consider only the entries $M_{k-1,\ell}$, where $\ell \geq F_{k,\lceil \frac{N+1}{2} \rceil}$. In general, one can show that in stage t, to compute the entries $M_{k,j}$ with $j = \lceil \frac{2i-1}{2^t}(N+1) \rceil$ and $1 \leq i \leq 2^{t-1}$, only the entries $M_{k-1,\ell}$ must be considered, where $F_{k,l} \leq \ell \leq F_{k,r}$ and l and r are $\lceil \frac{i-1}{2^{t-1}}(N+1) \rceil$ and $\lceil \frac{i}{2^{t-1}}(N+1) \rceil$, respectively. Notice that these entries have been computed in earlier stages. The above process repeats for every row of the matrix. The algorithm proceeds till the last entry $M_{K,N}$, the required optimal cost, is computed. The strict leftmost optimal solution $SLMO_{N,K} = (B_1, B_2, \dots, B_{K-1})$ is obtained, where $B_{k-1} = F_{k,B_k}$ for $1 < k \leq K$ and $B_K = N$.

As regard to the time complexity, first note that the total number of comparisons involved in a stage of the Dichotomic algorithm is $O(N)$ since it is equal to the sum of the number of values the variable ℓ takes in Loop 3, that is:

$$\sum_{i=1}^{2^{t-1}} (F_{k,\left\lceil \frac{i}{2^{t-1}}(N+1) \right\rceil} - F_{k,\left\lceil \frac{i-1}{2^{t-1}}(N+1) \right\rceil} + 1) = F_{k,N+1} - F_{k,0} + 2^{t-1} = N - 1 + 2^{t-1} = O(N)$$

Since Loop 2 runs $\lceil \log N \rceil$ times and Loop 1 is repeated K times, the overall time complexity is $O(NK \log N)$.

73.3.3 Two Channels

This subsection exploits the structure of the optimal solution in the special case where the item lengths are uniform and there are only two channels. Indeed, as shown later, the values assumed varying ℓ in the right-hand side of recurrence (73.2) for $k = 2$ form a *unimodal* sequence. That is, there is a particular index ℓ such that the values on its left-hand side are in nonincreasing order, while those on its right-hand side are in increasing order. By this fact, one can search the minimum of recurrence (73.2) in a very effective way, improving on the overall running time.

Formally, the 2-Uniform Allocation Problem consists in finding a partition S into two groups G_1 and G_2 such that $AED_S = \frac{1}{2}(N_1 P_1 + N_2 P_2)$ is minimized. Clearly, $N = N_1 + N_2$, and by Lemma 73.1, $N_1 \leq N_2$ holds for any optimal solution. Moreover, recall that any feasible segmentation S for $K = 2$ can be denoted by the single border B_1, which coincides with N_1.

Lemma 73.3 (Anticaglia et al. [14])

Consider the uniform length items $d_1, d_2, \dots, d_N$ sorted by nonincreasing probabilities, and $K = 2$ channels. Let $S = (N_1)$ be a feasible segmentation such that $P_1 \leq P_2$. If the segmentation $S' = (N_1 + 1)$ is feasible, then $AED_{S'} \leq AED_S$.

While Lemma 73.1 gives the upper bound $N_1 \leq \lfloor \frac{N}{2} \rfloor$ on the cardinality of group G_1, Lemma 73.3 provides a lower bound b on N_1. Indeed, it guarantees that any optimal solution contains at least the first b items $d_1, \dots, d_b$, where b is the largest index for which $P_1 = \sum_{h=1}^{b} p_h \leq P_2 = \sum_{h=b+1}^{N} p_h$. Formally, recurrence (73.2) for $K = 2$ can be rewritten as follows:

$$opt_{N,2} = \min_{b \leq \ell \leq \lfloor \frac{N}{2} \rfloor} \{C_{1,\ell} + C_{\ell+1,N}\} \tag{73.5}$$

where

$$b = \max_{1 \leq s \leq \lfloor \frac{N}{2} \rfloor} \left\{ s : \sum_{h=1}^{s} p_h \leq \sum_{h=s+1}^{N} p_h \right\}$$

```
Procedure BinSearch $(i, j)$;
    $m \leftarrow \lfloor \frac{i+j}{2} \rfloor$;
    if $i = j$ then
        return $m$
    else
        if $f(m) \geq f(m+1)$ then
            BinSearch $(m+1, j)$
        else
            BinSearch $(i, m)$;
```

FIGURE 73.2 The binary search on a unimodal sequence.

The following lemma improves on the upper bound of N_1 given by Lemma 73.1, and shows that the values of the feasible segmentations assumed in the right-hand side of Eq. (73.5) form a unimodal sequence.

Lemma 73.4 (Anticaglia et al. [14])

Consider the uniform length items $d_1, d_2, \ldots, d_N$ sorted by nonincreasing probabilities, and $K = 2$ channels. Let $S = (N_1)$ be a feasible solution such that $P_1 > P_2$. Consider the solutions $S' = (N_1{+}1)$ and $S'' = (N_1{+}2)$. If $AED_{S'} > AED_S$, then $AED_{S''} > AED_{S'}$.

In practice, one can scan the feasible solutions of Eq. (73.5) by moving the border ℓ rightwards, one position at a time, starting from the lower bound b obtained applying Lemma 73.3. The scan continues while the AED of the current solution does not increase, but stops as soon as the AED starts to increase. Indeed, by Lemma 73.4, further moving the border ℓ to the right can only increase the cost of the solutions. Hence the border m that minimizes Eq. (73.5), that is, the optimal solution of the problem, is given by

$$opt_{N,2} = C_{1,m} + C_{m+1,N} \tag{73.6}$$

where

$$m = \min_{b \leq \ell \leq \lfloor \frac{N}{2} \rfloor} \left\{ \ell \,:\, C_{1,\ell} + C_{\ell+1,N} < C_{1,\ell+1} + C_{\ell+2,N} \right\}$$

Note that, in the above equation, the cost variation is

$$(C_{1,\ell+1} + C_{\ell+2,N}) - (C_{1,\ell} + C_{\ell+1,N}) = \frac{1}{2}\left(\sum_{h=1}^{\ell} p_h - \sum_{h=\ell+1}^{N} p_h + p_{\ell+1}(2\ell + 2 - N) \right)$$

Due to the unimodal property of the sequence of values on the right-hand side of Eq. (73.6), the search of m can be done in $O(\log N)$ time by a suitable modified binary search. Let $f(\ell) = C_{1,\ell} + C_{\ell+1,N} = \frac{\ell}{2}\sum_{h=1}^{\ell} p_h + \frac{N-\ell}{2}\sum_{h=\ell+1}^{N} p_h$. Then, the unimodal sequence consists of the values $f(b)$, $f(b+1), \ldots,$ $f(\lfloor \frac{N}{2} \rfloor)$. As said, solving Eq. (73.6) is equivalent to find the index m such that $f(b) \geq \cdots \geq f(m) <$ $f(m+1) < \cdots < f(\lfloor \frac{N}{2} \rfloor)$. This can be done by invoking the recursive procedure *BinSearch*, given in Figure 73.2, with parameters $i = b$ and $j = \lfloor \frac{N}{2} \rfloor$. The BinSearch procedure first computes the middle point $m = \lfloor \frac{i+j}{2} \rfloor$. Then, the values $f(m)$ and $f(m{+}1)$ are compared in the light of the unimodal sequence definition. If $f(m) \geq f(m+1)$, the minimum must belong to the right half, otherwise it must be in the left half. Procedure BinSearch proceeds recursively on the proper half until the minimum is reached.

73.4 Nonuniform Lengths

Consider now the K-Nonuniform Allocation Problem for an arbitrary number K of channels. In contrast to the uniform case, introducing items with different lengths makes the problem computationally intractable.

Theorem 73.1 (Ardizzoni et al. [1])

The K-Nonuniform Allocation Problem is NP-hard for $K = 2$, and strong NP-hard for an arbitrary K.

As a consequence of the above result, there is no pseudopolynomial time optimal algorithm or fully polynomial time approximation scheme (FPTAS) for solving the K-Nonuniform Allocation Problem (unless P=NP). However, when the maximum item length z is bounded by a constant, a polynomial time optimal algorithm can be derived where z appears in the exponent. When $z = 1$, this algorithm reduces to the DP algorithm. The following result generalizes Lemma 73.1.

Lemma 73.5 (**Ardizzoni et al.** [**1**])

Let G_h and G_j be two groups in an optimal solution for the K-Nonuniform Allocation Problem. Let d_i and d_k be items with $z_i = z_k$ and $d_i \in G_h$, $d_k \in G_j$. If $Z_h < Z_j$, then $p_i \geq p_k$. Similarly, if $p_i > p_k$, then $Z_h \leq Z_j$.

Based on the above lemma, some additional notations are introduced. The set D of items can be viewed as a union of disjoint subsets $D_i = \{d_1^i, d_2^i, \ldots, d_{L_i}^i\}$, $1 \leq i \leq z$, where D_i is the set of items with length i, L_i is the cardinality of D_i, and z is the maximum item length. Let p_j^i represent the probability of item d_j^i, for $1 \leq j \leq L_i$.

The following corollary generalizes Corollary 73.1.

Corollary 73.3 (**Ardizzoni et al.** [**1**])

Let $d_1^i, d_2^i, \ldots, d_{L_i}^i$ be the L_i items of length i with $p_m^i \geq p_n^i$ whenever $m < n$, for $i = 1, \ldots, z$. There is an optimal solution for partitioning the items of D into K groups $G_1, \ldots, G_K$, such that if $a < b < c$ and $d_a^i, d_c^i \in G_j$, then $d_b^i \in G_j$.

In the rest of this section, the items in each D_i are assumed to be sorted by nonincreasing probabilities, and optimal solutions will be sought of the form:

$$\underbrace{d_1^1, \ldots, d^1_{B_1^{(1)}}}_{G_1}, \underbrace{d^1_{B_1^{(1)}+1}, \ldots, d^1_{B_2^{(1)}}}_{G_2}, \ldots, \underbrace{d^1_{B_{K-1}^{(1)}+1}, \ldots, d^1_{N_1}}_{G_K}$$

$$\underbrace{d_1^2, \ldots, d^2_{B_1^{(2)}}}_{G_1}, \underbrace{d^2_{B_1^{(2)}+1}, \ldots, d^2_{B_2^{(2)}}}_{G_2}, \ldots, \underbrace{d^2_{B_{K-1}^{(2)}+1}, \ldots, d^2_{N_2}}_{G_K}$$

$$\vdots$$

$$\underbrace{d_1^z, \ldots, d^z_{B_1^{(z)}}}_{G_1}, \underbrace{d^z_{B_1^{(z)}+1}, \ldots, d^z_{B_2^{(z)}}}_{G_2}, \ldots, \underbrace{d^z_{B_{K-1}^{(z)}+1}, \ldots, d^z_{N_z}}_{G_K}$$

where $B_j^{(i)}$ is the highest index among all items of length i in group G_j. The solution will be represented as $(\bar{B}_1, \bar{B}_2, \ldots, \bar{B}_{K-1})$, where each $\bar{B}_j$ is the z-tuple $(B_j^{(1)}, B_j^{(2)}, \ldots, B_j^{(z)})$ for $1 \leq j \leq K-1$. From now on, $B_{K-1}^{(i)}$ will be referred to as the *final border for length i* and $\bar{B}_{K-1}$ as the *final border vector.*

Let $OPT_{n_1,\ldots,n_z,k}$ denote the optimal solution for grouping the $\sum_{i=1}^{z} n_i$ items $d_1^i, d_2^i, \ldots, d_{n_i}^i$, $1 \leq i \leq z$, into k groups and let $opt_{n_1,\ldots,n_z,k}$ be its corresponding cost. Let $C_{l_1,n_1,\ldots,l_z,n_z}$ be the cost of putting items l_i through n_i, for all $i = 1, 2, \ldots, z$, into one group, that is,

$$C_{l_1,n_1,\ldots,l_z,n_z} = \frac{1}{2}\left(\sum_{i=1}^{z} i(n_i - l_i + 1)\right)\left(\sum_{i=1}^{z}\sum_{j=l_i}^{n_i} p_j^i\right)$$

Now, consider the recurrence:

$$opt_{n_1,\ldots,n_z,k} = \min_{\substack{\bar{\ell}=(\ell_1,\ldots,\ell_z)\\ 0\leq \ell_i \leq n_i, 1\leq i\leq z}} \left\{ opt_{\ell_1,\ldots,\ell_z,k-1} + C_{\ell_1+1,n_1,\ldots,\ell_z+1,n_z} \right\} \tag{73.7}$$

To solve this recurrence by using dynamic programming, consider a $(z+1)$-dimensional matrix M, made of K rows in the first dimension and L_i columns in dimension $i+1$ for $i = 1, \ldots, z$. Each entry is represented by a $(z+1)$-tuple $M_{k,n_1,\ldots,n_z}$, where k corresponds to the row index and n_i to the index of the column in dimension $i+1$. The entry $M_{k,n_1,\ldots,n_z}$ represents the optimal cost for partitioning items d_1^i through $d_{n_i}^i$, for $i = 1, 2, \ldots, z$, into k groups. There is also a similar matrix F where the entry $F_{k,n_1,\ldots,n_z}$ corresponds to the final border vector of the solution whose cost is $M_{k,n_1,\ldots,n_z}$. The matrix entries are filled row by row. The optimal solution is given by $OPT_{L_1,\ldots,L_z,K} = (\bar{B}_1, \bar{B}_2, \ldots, \bar{B}_{K-1})$ where, starting from $\bar{B}_K = (L_1, L_2, \ldots, L_z)$, the value of $\bar{B}_k$ is obtained from the value of $\bar{B}_{k+1}$ and by F as $\bar{B}_k = F_{k+1,\bar{B}_{k+1}}$, for $k = 1, \ldots, K-1$. The Optimal algorithm derives directly from recurrence (73.7). Since the computation of every entry $M_{k,n_1,\ldots,n_z}$ and $F_{k,n_1,\ldots,n_z}$ requires $\prod_{i=1}^{z}(n_i+1) \leq \prod_{i=1}^{z}(L_i+1)$ comparisons, and every row has $\prod_{i=1}^{z} L_i$ entries, the overall time complexity is $O(K\prod_{i=1}^{z}(L_i+1)^2) = O(KN^{2z})$.

73.5 Heuristics

Since the K-Nonuniform Allocation Problem is strong NP-hard, it is computationally intractable unless P=NP. In practice, this implies that one is forced to abandon the search for efficient algorithms which find optimal solutions. Therefore, one can devise fast and simple heuristics that provide solutions which are not necessarily optimal but usually fairly close. This strategy is followed in this section, where the main heuristics are reviewed. Two of the heuristic methods are greedy and the third one is based on the dynamic programming procedure developed in earlier sections. An experimental evaluation of our heuristics is presented and the quality of the solutions generated is compared against a lower bound which is derived by relaxing the problem and then solving it optimally via the dynamic programming procedures developed in earlier sections. All heuristics assume that the items are sorted by nonincreasing $\frac{p_i}{z_i}$ ratios, which can be done in $O(N \log N)$ time during a preprocessing step.

73.5.1 The Greedy Algorithm

The Greedy heuristic [11,12] initially assigns all the N data items to a single group. Then, for $K-1$ times, one of the groups is split in two groups, that will be assigned to two different channels. To find which group to split along with its actual split point, all the possible points of all groups are considered as split point candidates, and the one that decreases AED the most is selected. In details, assume that the channel to be split contains the items from d_i to d_j, with $1 \leq i < j \leq N$, and let $cost_{i,j,2}$ denote the cost of a feasible solution for assigning such items to two channels. Then, the split point is the index m that satisfies

$$cost_{i,j,2} = C_{i,m} + C_{m+1,j} = \min_{i \leq \ell \leq j-1} \{C_{i,\ell} + C_{\ell+1,j}\} \tag{73.8}$$

An efficient implementation takes advantage from the fact that, between two subsequent splits, it is sufficient to recompute the costs for the split point candidates of the last group that has been actually split. The time complexity of the Greedy heuristic is $O(N(K + \log N))$ and $O(N \log N)$ in the worst and average cases, respectively [14].

Note that Greedy scales well when changes occur on the number of channels, on the number of items, on item probabilities, as well as on item lengths. Indeed, adding or removing a channel simply requires doing a new split or removing the last introduced split, respectively. Adding a new item first requires to insert such an item in the sorted item sequence. Assume the new item is added to group G_j, then the border of the two-channel subproblem including items of G_j and G_{j+1} is recomputed by applying Eq. (73.8). Similarly, deleting an item that belongs to group G_j requires to solve again the two-channel subproblem including items of G_j and G_{j+1}. Finally, a change in the probability/length of an item is equivalent to first removing that item and then adding the same properly modified item.

73.5.2 The Greedy+ Algorithm

The Greedy+ heuristic [14] is a refinement of the Greedy heuristic and consists of two phases. In the first phase, it behaves as Greedy, except the way the split point is determined. In the second phase, the solution provided by the first phase is refined by working on pairs of consecutive channels.

Specifically, in the first phase, Greedy+ uses an approach similar to that of Eq. (73.6) to determine the split point. This is because splitting one channel is the same as solving the problem for two channels. In details, the split point m is given by

$$cost_{i,j,2} = C_{i,m} + C_{m+1,j} \tag{73.9}$$

where

$$m = \min_{i \leq \ell \leq j-1} \left\{ \ell \, : C_{i,\ell} + C_{\ell+1,j} < C_{i,\ell+1} + C_{\ell+2,j} \right\}$$

Note that, since the item lengths are not the same, the sequence of values $C_{i,\ell} + C_{\ell+1,j}$, for $i \leq \ell \leq j-1$, is not unimodal. However, Greedy+ behaves as such a sequence were unimodal. Instead of trying all the possible values of ℓ between i and j, as done by Greedy, Greedy+ performs a left-to-right scan starting from i and stopping as soon the AED increases. In this way, a suboptimal solution $S = (B_1, B_2, \ldots . B_{K-1})$ is found.

The second phase is performed only when $K \geq 3$ and consists in refining the solution S by recomputing its borders. The phase consists in a sequence of odd steps, followed by a sequence of even steps. During the t-th odd step, $1 \leq t \leq \lfloor \frac{K}{2} \rfloor$, the two-channel subproblem including the items assigned to groups G_{2t-1} and G_{2t} is solved. Specifically, Eq. (73.9) is applied choosing $i = B_{2t-2} + 1$ and $j = B_{2t}$, thus recomputing the border B_{2t-1} of S. Similarly, during the t-th even step, $1 \leq t \leq \lfloor \frac{K-1}{2} \rfloor$, the two-channel subproblem including the items assigned to groups G_{2t} and G_{2t+1} is solved by applying Eq. (73.9) with $i = B_{2t-1} + 1$ and $j = B_{2t+1}$, recomputing the border B_{2t} of S.

The initial sorting requires $O(N \log N)$ time. Since each split runs in $O(N)$ time, and K splits are computed, the first phase of Greedy+ takes $O(NK)$ time. The second phase of Greedy+ requires $O(N)$ time since each item is considered as a candidate split point at most in a single split computation among all the odd steps, and in a single split computation among the even steps. Therefore, the overall time required in the worst case by the Greedy+ heuristic is $O(N(K + \log N))$. Clearly, Greedy+ maintains the same scaling features as Greedy.

73.5.3 The Dlinear Algorithm

The Dlinear heuristic [14] follows a dynamic programming approach similar to that provided by recurrence (73.2). Fixed k and n, Dlinear computes a solution for n items from the previously computed solution for $n - 1$ items and k channels, exploiting the characteristics of the optimal solutions for two channels and uniform lengths.

Let $M_{k,n}$ and $F_{k,n}$ be defined as in Section 73.3.1. Dlinear selects the feasible solutions that satisfy the following Recurrence:

$$M_{k,n} = \begin{cases} C_{1,n} & \text{if } k = 1 \\ M_{k-1,m} + C_{m+1,n} & \text{if } k > 1 \end{cases} \tag{73.10}$$

where

$$m = \min_{F_{k,n-1} \leq \ell \leq n-1} \left\{ \ell \, : M_{k-1,\ell} + C_{\ell+1,n} < M_{k-1,\ell+1} + C_{\ell+2,n} \right\}$$

In practice, Dlinear pretends to adapt Eq. (73.6), that holds for the 2-Uniform Allocation Problem, also to the K-Nonuniform Allocation Problem. In particular, the choice of the lower bound $F_{k,n-1}$ in the formula of m is suggested by Lemma 73.2, which says that the border of channel $k - 1$ can only move right when a new item with the smallest probability is added. Moreover, m is determined as in Eq. (73.6) pretending that the sequence $M_{k-1,\ell} + C_{\ell+1,n}$, obtained for $F_{k,n-1} \leq \ell \leq n-1$, be unimodal. Therefore, the solution provided by Dlinear is a suboptimal one.

As regard to the time complexity, computing $M_{k,n}$ requires $O(F_{k,n} - F_{k,n-1})$ time. Hence, row k of M is filled in $\sum_{n=1}^{N} O(F_{k,n} - F_{k,n-1}) = O(F_{k,N} - F_{k,1}) = O(N)$ time. Since M has K rows and the sorting step takes $O(N \log N)$ time, the overall time complexity of the Dlinear algorithm is $O(N(K + \log N))$.

73.6 Experimental Tests

In this section, the behavior of the Greedy, Greedy+, and Dlinear heuristics is tested. The algorithms are written in C and the experiments are run on an AMD Athlon XP 2500+, 1.84 GHz, with 1 GB RAM.

The heuristics are executed on the following nonuniform length instances. Given the number N of items and a real number $0 \le \theta \le 1$, the item probabilities are generated according to a Zipf distribution whose skew is θ, namely:

$$p_i = \frac{(1/i)^{\theta}}{\sum_{h=1}^{N}(1/h)^{\theta}}, \quad 1 \le i \le N$$

In the above formula, $\theta = 0$ stands for a uniform distribution with $p_i = \frac{1}{N}$, while $\theta = 1$ implies a high skew, namely the range of p_i values becomes larger. The item lengths z_i are integers randomly generated according to a uniform distribution in the range $1 \le z_i \le z$. The items are sorted by nonincreasing $\frac{p_i}{z_i}$ ratios. The parameters N, K, z, and θ vary, respectively, in the ranges: $500 \le N \le 2500$, $10 \le K \le 500$, $3 \le z \le 10$, and $0.5 \le \theta \le 1$.

Since the Optimal algorithm can find the exact solutions in a reasonable time only for small instances, a *lower bound* on AED is used for large values of N, K, and z. The lower bound for a nonuniform instance is obtained by transforming it into a uniform instance as follows. Each item d_i of probability p_i and length z_i is decomposed in z_i items of probability $\frac{p_i}{z_i}$ and length 1. Since more freedom has been introduced, it is clear that the optimal AED for the so transformed problem is a lower bound on the AED of the original problem. Since the transformed problem has uniform lengths, its optimal AED is obtained by running the Dichotomic algorithm.

The simulation results are exhibited in Tables 73.1–73.4. The tables report the time (measured in microseconds), the AED, and the percentage of error, which is computed as

$$\left(\frac{\text{AED}_{\text{heuristic}} - \text{AED}_{\text{lowerbound}}}{\text{AED}_{\text{lowerbound}}}\right) 100$$

The running times reported in the tables do not include the time for sorting.

By observing the tables, one notes that Greedy+ and Dlinear always outperform Greedy in terms of solution quality. In particular, Greedy+ at least halves the error of Greedy, producing solutions whose errors are at most 5.7%. Moreover, Dlinear reaches the optimum almost in all cases, and its maximum error is as high as 1.8% only in one instance. As regard to the running times, although all the three heuristics have the same asymptotic worst-case time, Greedy is the fastest in practice. Although Greedy+ and Dlinear are slower than Greedy, their running times are always less than one tenth of second. The experiments show that Greedy+ and Dlinear behave well when the item probabilities follow a Zipf distribution. This suggests that, in most cases, the AED achieved in correspondence of the leftmost value of ℓ satisfying recurrences (73.9) and (73.10) is the optimal AED or it is very close to the optimal AED. In other words, the sequence of values obtained by varying ℓ is almost unimodal.

In summary, the experimental tests show that the Dlinear heuristic finds optimal solutions almost always. In contrast, Greedy is the fastest heuristic, but produces the worst solutions. Finally, Greedy+ presents running times and suboptimal solutions, which are both intermediate between those of Greedy and Dlinear. Therefore, the choice among the heuristics depends on the goal to be pursued. If one is interested in finding the best suboptimal solutions, then Dlinear should be adopted. Instead, if the running time is the main concern, then Greedy should be chosen, while if adaptability to parameter changes is the priority, then either Greedy or Greedy+ could be applied. In this scenario, Greedy+ represents a good compromise since it is scalable and produces fairly good solutions.

TABLE 73.1 Experimental Results When $K = 20$, $\theta = 0.8$, and $z = 3$

$N/K/\theta/z$	Algorithm	AED	% Error	Time
500/20/0.8/3	Greedy	18.72	7.1	102
	Greedy+	17.58	0.6	3514
	Dlinear	17.47		2106
	Lower bound	17.47		
1500/20/0.8/3	Greedy	53.85	7.9	283
	Greedy+	51.71	3.6	21240
	Dlinear	49.90		6519
	Lower bound	49.90		
1750/20/0.8/3	Greedy	62.64	7.9	326
	Greedy+	58.92	1.5	31137
	Dlinear	58.04		7488
	Lower bound	58.04		
2000/20/0.8/3	Greedy	71.24	7.9	373
	Greedy+	66.93	1.4	38570
	Dlinear	65.98		8602
	Lower bound	65.98		
2250/20/0.8/3	Greedy	79.70	7.8	457
	Greedy+	75.06	1.6	45170
	Dlinear	73.87		9749
	Lower bound	73.87		
2500/20/0.8/3	Greedy	88.40	7.8	474
	Greedy+	82.51	0.7	62376
	Dlinear	81.93		10920
	Lower bound	81.93		

TABLE 73.2 Experimental Results When $N = 2500$, $\theta = 0.8$, and $z = 3$

$N/K/\theta/z$	Algorithm	AED	% Error	Time
2500/10/0.8/3	Greedy	179.16	7.8	381
	Greedy+	167.86	1.0	97356
	Dlinear	166.14		4919
	Lower bound	166.14		
2500/40/0.8/3	Greedy	44.04	7.9	562
	Greedy+	41.58	1.9	34147
	Dlinear	40.79		22771
	Lower bound	40.79		
2500/80/0.8/3	Greedy	21.98	7.9	685
	Greedy+	20.72	1.7	19179
	Dlinear	20.37		46545
	Lower bound	20.37		
2500/100/0.8/3	Greedy	17.14	5.2	740
	Greedy+	16.75	2.8	27452
	Dlinear	16.29		57906
	Lower bound	16.29		
2500/200/0.8/3	Greedy	8.56	5.1	1009
	Greedy+	8.37	2.8	12974
	Dlinear	8.15	0.1	116265
	Lower bound	8.14		
2500/500/0.8/3	Greedy	3.4	4.2	2313
	Greedy+	3.35	2.7	21430
	Dlinear	3.32	1.8	273048
	Lower bound	3.26		

TABLE 73.3 Experimental Results When $N = 2500$, $K = 50$, and $z = 3$

$N/K/\theta/z$	Algorithm	AED	% Error	Time
2500/50/0.5/3	Greedy	47.74	9.7	595
	Greedy+	46.02	5.7	23175
	Dlinear	43.52	0.02	29075
	Lower bound	43.51		
2500/50/0.7/3	Greedy	39.59	6.8	600
	Greedy+	38.47	3.8	23606
	Dlinear	37.05	0.02	29132
	Lower bound	37.04		
2500/50/0.8/3	Greedy	34.33	5.2	603
	Greedy+	33.49	2.6	24227
	Dlinear	32.61		29121
	Lower bound	32.61		
2500/50/1/3	Greedy	23.10	3.2	609
	Greedy+	22.53	0.6	27566
	Dlinear	22.38		28693
	Lower bound	22.38		

TABLE 73.4 Experimental Results When $N = 500$, $K = 50$, and $\theta = 0.8$

$N/K/\theta/z$	Algorithm	AED	% Error	Time
500/50/0.8/3	Greedy	7.34	5.3	147
	Greedy+	7.19	3.1	2517
	Dlinear	6.98	0.1	5423
	Lower bound	6.97		
500/50/0.8/5	Greedy	10.78	5.3	147
	Greedy+	10.52	2.8	2938
	Dlinear	10.25	0.1	5490
	Lower bound	10.23		
500/50/0.8/7	Greedy	14.50	4.9	146
	Greedy+	14.16	2.4	3329
	Dlinear	13.85	0.2	5499
	Lower bound	13.82		
500/50/0.8/10	Greedy	19.48	5.1	145
	Greedy+	18.97	2.3	3899
	Dlinear	18.58	0.2	5507
	Lower bound	18.53		

73.7 Conclusions

In this chapter, the problem of data broadcasting over multiple channels, with the objective of minimizing the AED of the clients, was considered under the assumptions of skewed allocation to multiple channels and flat scheduling per channel. Both the uniform and nonuniform length problems were solved to the optimum, illustrating exact algorithms based on dynamic programming. Moreover, effective heuristics for nonuniform lengths have also been shown. All the results reviewed in this chapter are summarized in Table 73.5.

In this chapter, the client delay has been defined as the overall time elapsed from the moment the client desires a data item to the moment the item download starts. Such a definition assumes that indexing is already available to the client. Hence, the client delay does not include the tuning time spent by the client for actively retrieving the index information and the data item. Thus, after reading the index, the client can be turned into a power saving mode until the data item appears on the proper channel. Therefore, our solution minimizes the AED and keeps as low as possible the tuning time provided that an efficient index

TABLE 73.5 Results for Broadcasting N Data Items on K Channels with Skewed Allocation and Flat Scheduling

Item Lengths	Complexity	Solution	Algorithm	Time	References
Uniform	P	Optimal	DP	$O(KN^2)$	[12]
		Optimal	Dichotomic	$O(KN \log N)$	[1]
Nonuniform	Strong	Optimal	Optimal	$O(KN^{2z})$	[1]
	NP-hard	Heuristic	Greedy, Greedy+, Dlinear	$O(N(K + \log N))$	[11,14]

strategy is adopted on one or more separate channels. In our solution, the index can be readily derived from the $(K - 1)$-tuple $(B_1, B_2, \ldots, B_{K-1})$, which compactly represents the data allocation. However, this tuple is enough for indexing only if all the clients know, as a global information, the relative position of each data item within the set of all data items sorted by probabilities. To overcome this assumption, solutions can be sought that, without using global information on data items, either mix index and data items within the same channels or optimize the index broadcasting on dedicated channels [16,17].

Acknowledgment

Supported by ISTI-CNR under the BREW research grant.

References

[1] Ardizzoni, E., Bertossi, A. A., Pinotti, M. C., Ramaprasad, S., Rizzi, R., and Shashanka, M. V. S., Optimal skewed data allocation on multiple channels with flat broadcast per channel, *IEEE Trans. Comput.*, 54(5), 558, 2005.

[2] Ammar, M. H. and Wong, J. W., The design of teletext broadcast cycles, *Perform Evaluation*, 5(4), 235, 1985.

[3] Ammar, M. H. and Wong, J. W., On the optimality of cyclic transmission in teletext systems, *IEEE Trans. Commn.*, 35(11), 1159, 1987.

[4] Bar-Noy, A., Bhatia, R., Naor, J. S., and Schieber, B. Minimizing service and operation costs of periodic scheduling, *Proc. 9th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 1998, p. 11.

[5] Imielinski, T., Viswanathan, S., and Badrinath, B. R., Energy efficient indexing on air, *Proc. SIGMOD*, Minneapolis, May 1994, pp. 25–36.

[6] Kenyon, C. and Schabanel, N., The data broadcast problem with non-uniform transmission time, In *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 1999, p. 547.

[7] Kenyon, C., Schabanel, N., and Young, N., Polynomial time approximation scheme for data broadcast, *Proc. ACM Symp. on Theory of Computing (STOC)*, 2000, p. 659.

[8] Peng, W. C. and Chen, M. S., Efficient channel allocation tree generation for data broadcasting in a mobile computing environment, *Wireless Networks*, 9(2), 117, 2003.

[9] Prabhakara, K. A., Hua, K. A., and Oh, J., Multi-level multi-channel air cache designs for broadcasting in a mobile environment, *Proc. 16th IEEE Int. Conf. on Data Engineering (ICDE)*, San Diego, February 2000, pp. 167–176.

[10] Vaidya, N. and Hameed, S., Log time algorithms for scheduling single and multiple channel data broadcast, *Proc. 3rd ACM-IEEE Conf. on Mobile Computing and Networking (MOBICOM)*, Budapest, September 1997, pp. 90–99.

[11] Yee, W. G., Efficient Data Allocation for Broadcast Disk Arrays, Technical report, GIT-CC-02-20, Georgia Institute of Technology, 2001.

[12] Yee, W. G., Navathe, S., Omiecinski, E., and Jermaine, C., Efficient data allocation over multiple channels at broadcast servers, *IEEE Trans. Comput.*, 51(10), 1231, 2002.

[13] Acharya, S., Alonso, R., Franklin, M., and Zdonik, S., Broadcast disks: data management for asymmetric communication environments, *Proc. SIGMOD*, San Jose, May 1995, pp. 199–210.

[14] Anticaglia, S., Barsi, F., Bertossi, A. A., Iamele, L., and Pinotti, M. C., Efficient Heuristics for Data Broadcasting on Multiple Channels, Technical report, 2005/5, Department of Mathematics and Computer Science, University of Perugia, 2005. Published online Wireless Networks, on October 12th 2006.

[15] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S., Web caching and Zipf-like distributions: evidence and implications, *Proc. IEEE INFOCOM*, New York, 1999, pp. 126–134.

[16] Lo, S.-C. and Chen, A. L. P., Optimal index and data allocation in multiple broadcast channels, *Proc. 16th IEEE Int. Conf. on Data Engineering (ICDE)*, San Diego, February 2000, pp. 293–302.

[17] Stojmenovic, I., Ed., *Handbook of Wireless Networks and Mobile Computing*, Wiley, Chichester, 2002.

74

Combinatorial and Algorithmic Issues for Microarray Analysis

Carlos Cotta
University of Málaga

Michael A. Langston
University of Tennessee

Pablo Moscato
The University of Newcastle

74.1 Introduction

J. Craig Venter declared in 1998: "We are now starting the Century of Biology." This had been recognized before. In fact Gregory Benford was already pointing to this fact in 1995, when he noted that physics had dominated the twentieth century, as chemistry had probably dominated the century before. In his own words:

> And yet, far from the physics departments of the great campuses, a clarion call is sounding through our time, one that responds to hot-button environmental problems and that incorporates rapid advances in other laboratories: Biology has turned aggressively useful.

Microarrays have been evolving rapidly, and are among the most novel and revolutionary new biotechnologies. They are reshaping our understanding of biological systems as well as shaking the grounds of biomedical research. They allow us to monitor the expression of thousands of genes at once. With a single experiment we can test billions of individual hypotheses. A query at PubMed[1] shows that over 20,000 publications already have the words "microarray" or "DNA array." Almost all of these papers have appeared in the last 10 years, with approximately 70% of them appearing in the last two years. These high-throughput molecular assays generate immense datasets, which have the potential to help us to understand biological systems in ways that are completely new. While huge promises are ritually proclaimed

[1] http://www.ncbi.nlm.nih.gov/entrez/query.fcgi

(personalized medicine, targeted therapies, genetic engineering for more efficient crops, etc.) [1], the challenges are equally enormous [2].

Advances in combinatorial optimization are too moving along swiftly. Fixed-parameter tractable algorithms, for example, speed the systematic development of data reduction methods to bound the search for optimal solutions. As another example, metaheuristics produce powerful stochastic algorithms for large-scale optimization problems. In spite of this progress, combinatorial optimization has sometimes been rather naively applied. The ultimate goal is not always to find a purely optimal solution, but instead to use results in the context of other tools to uncover underlying genetic networks or other conserved aspects of biosystems.

In this chapter, we present three illustrative examples drawn from the authors' own experience in the analysis of microarray datasets. The associated decision problems are each $\mathcal{NP}$-complete. An underlying theme is subgraph identification via cliques, bicliques, and Hamiltonian paths.

74.2 Genetic Networks and the Clique Problem

The structure of a biological network has a natural representation as a graph. As a consequence, algorithms for optimal subgraph detection become powerful tools for the investigation of biological function. In gene regulatory networks, any given gene may have different functions as its activity influences, and is influenced by, a number of other genes [3]. A gene in one species may be very similar, at the sequence level, to another gene in some other species. Given such "orthologs," common subgraphs among different biological networks help us infer evolutionarily conserved modules of coexpressed genes [4–6]. This leads to approaches for deriving phylogenetic trees based on the detection of common metabolic pathways between taxa [7]. Biology, thus aided by graph-theoretic formulations, is now moving from the study of single genes and proteins to the investigation of the basic common building blocks of life.

At the core of this quest is the search for sets of putatively coregulated genes. This can be formalized as the CLIQUE problem (see Ref. [8] for an excellent review of this foundational problem). Formally, the inputs to CLIQUE are an undirected graph G with n vertices, and a parameter $k \leq n$. The decision problem asks whether G contains a clique of size k, that is, a subgraph isomorphic to K_k, the complete graph on k vertices. Because CLIQUE is $\mathcal{NP}$-complete, it has no known decision algorithm that runs in time polynomial in the size of the input. CLIQUE can be decided by generating and checking all $\binom{n}{k}$ of vertices selections. But this brute force approach requires $O(n^k)$ time, and is thus prohibitively slow, even for problem instances of only modest size.

Despite its computational intractability, there is a strategic advantage to formulating problems of biological interest in terms of cliques. This is because a vertex can be a member of multiple cliques, just as genes and gene products can be involved in multiple pathways. Traditional clustering algorithms, however, are limited by the requirement that a vertex must reside in a single cluster [9–13]. A few clustering techniques, for example, those employing factor analysis, do not require exclusive cluster membership for single genes [14]. Unfortunately, these tend to produce biologically uninterpretable factors without the incorporation of prior biological information [15]. CLIQUE has no such restriction.

74.3 Parameterized Complexity

74.3.1 Fixed-Parameter Tractability

The origins of *fixed-parameter tractability* (FPT) can be traced at least as far back as work on the applications of well-quasi order theory. Nearly two decades ago, it was shown that a variety of $\mathcal{NP}$-complete problems are tractable when a relevant input parameter is fixed [16,17]. A problem is FPT if it can be solved in $O(f(k)n^c)$ time, where n is the size of the instance, k the input parameter, and c a constant independent of both n and k [18].

A number of problems of interest in bioinformatics are FPT. One of the most prominent ones is VERTEX COVER. Here the input is an undirected graph G with n vertices, and a parameter $k \leq n$. The decision problem asks whether G contains a set C of k or fewer vertices that covers every edge in G (an edge is said to be covered if either one or both of its endpoints are in C).

Alas, CLIQUE is not FPT unless the $\mathcal{W}$ hierarchy collapses [18]. (The $\mathcal{W}$ hierarchy, whose lowest level is FPT, can be viewed as a fixed-parameter analog of the polynomial hierarchy, whose lowest level is $\mathcal{P}$.) Fortunately, CLIQUE's complementary dual is VERTEX COVER. To see this, let $\overline{G}$ denote the complement of G. ($\overline{G}$ has the same vertex set as G, and all the edges present in G are absent in $\overline{G}$ and vice versa.) A vertex cover of size k in $\overline{G}$ turns out to be exactly the complement of a clique of size $n - k$ in G. Thus the search for a minimum vertex cover in $\overline{G}$ corresponds to the search for a maximum clique in G.

74.3.2 Kernelization and Branching

An effective approach to finding a small vertex cover is accomplished with *kernelization* and *branching*. We start by reducing an arbitrary input instance to what is hopefully a much smaller instance (the *kernel*). It is easy to see that an $O(k^2)$ kernel suffices [19]. Kernels of size $O(k)$ can also be obtained at the expense of methods relying on linear programming relaxation [20,21], which tend to be slower in practice.

More recently, a new technique, termed *crown reduction*, was introduced for kernelization. A *crown* is an ordered pair (I, H) of subsets of vertices from G that satisfies the following criteria: (1) $I \neq \emptyset$ *is an independent set of* G, (2) $H = N(I)$, *and* (3) there exists a matching M on the edges connecting I and H such that all elements of H are matched. H is called the *head* of the crown. The *width* of the crown is $|H|$.

The following theorem is then central to this algorithmic approach.

Theorem 74.1 (Abu-Khzam et al. [22])

Any graph G can be decomposed into a crown (I, H) for which H contains a minimum size vertex cover of G and so that $|H| \leq 3k$. Moreover, the decomposition can be accomplished in $O(n^{\frac{5}{2}})$ time.

Branching is applied after the kernel is obtained. A binary tree search is used. Subtree searches can be spawned off at each level and be concurrently explored [23]. Up to 64 processors have been used for an application in motif discovery [24]. Contrary to the folklore of $\mathcal{NP}$-completeness, this method can be used to solve huge instances optimally [25]. For large problems, and for particularly difficult subtrees, hardware acceleration in the form of Field Programmable Gate Arrays have delivered speedups in excess of 125 over software-only implementations [26].

74.3.3 The Clique Intersection Graph

To study high-level network structures we need first to enumerate all maximal cliques of the graph under study. Note that a graph may have as many as $3^{n/3}$ maximal cliques. Thus memory management is critical. Highly efficient methods are discussed in Ref. [27].

To illustrate, we recently analyzed a dataset with 6,830 genes (more on this in the sequel). A threshold of 0.85 was chosen for the minimum meaningful correlation between gene pairs. This cutoff produced a graph with only 2,281 vertices and 2,619 edges. It contained 355 maximal (locally optimal) cliques with sizes between 3 and 15 vertices. From this we computed the clique interaction graph defined as follows: A vertex in the clique interaction graph denotes a maximal clique in the original graph; two vertices in the clique interaction graph are connected by an edge if and only if the corresponding cliques are not disjoint. See Figure 74.1.

74.4 A Biclique-Oriented Approach

We have mentioned before how an approach based on clique finding and new techniques based on FPT have been useful to identify highly correlated groups of genes. In some cases, however, the different samples belong to particular classes of interest to the biologist or the medical researcher. The samples can correspond

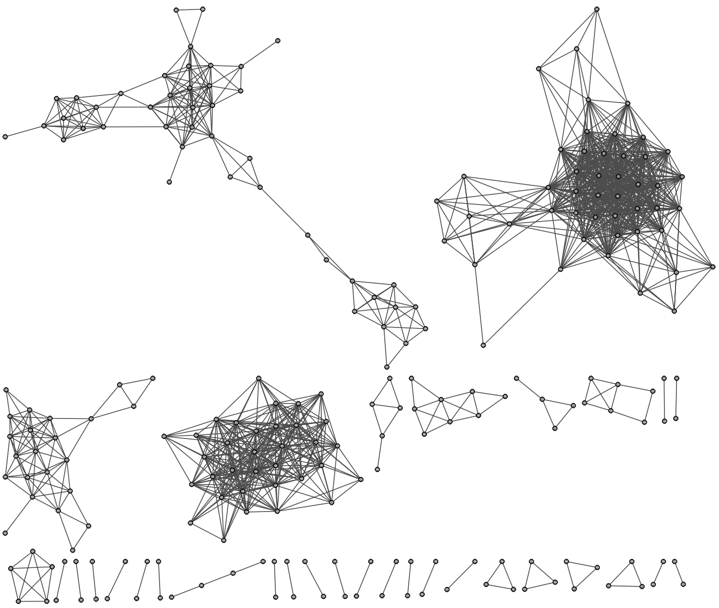


FIGURE 74.1 The clique intersection graph obtained in this study. A graph is first constructed with 6,830 vertices (in a one-to-one correspondence with all the different genes in the original microarray). Edges in this graph links pairs of genes that have a correlation greater than 0.85 or smaller than −0.85. We then calculated its clique intersection graph, shown in this figure. This graph, in conjunction with the genetic signatures found with the $(\alpha, \beta) - k$-feature set method allows to identify differential pathways associated with the disease. For instance, the K_5 at the bottom-left corner represents a set of cliques entirely composed of genes present in the Colon genetic signature shown in Figure 74.3(c) The RPS16 gene (Ribosomal Protein S16) is present in all five cliques in the original graph, it is highly expressed in five cell lines, significantly less but still expressed in HCT-116 and underexpressed in HCT-15, matching recent reports [28]). Another gene common to all cliques is IL20RA, which encodes for receptor for interleukin 20 (IL20), a cytokine that may be involved in epidermal function. IL20RA is highly expressed in skin, upregulated in psoriasis, and may have an important role in local mechanisms of mucosal and cutaneous immunity [29]. Our combinatorial methods allow a systematic investigation of what can be "master genes" as being key players in a variety of pathways implicated in the disease and allow for high-throughput bioinformatic analysis.

to either particular clearly separated clinical conditions [30], or to different cellular processes [31,32], to different parts of an organ (voxelization techniques) [33], different cancer types [34], prediction of tumor outcome [35], different cell lines [36], etc. Now the question is: *Given that such a labeling on the samples is available, can we identify the set of genes that most likely explains the existence of these classes?*

As such, this is a generic problem that needs a precise formalization. Since in a typical microarray experiment the number of samples is usually much smaller than the number of genes, it is often the case that several high correlations exist between some genes and the labeling. As a consequence, minimization of the number of genes that can "explain" the labeling should be taken with some caution. It would be possible that we can find a small number of genes for which the following holds. For any two pairs of samples that have different labelings it is always true that there exists at least one gene which has a significantly different expression value. As a consequence, we need to find some new formalization of this problem that would give "robust genetic signatures." By "robust," we mean that the explanation should rely on the coexpression of many genes, as a way of avoiding individual spurious correlations that may dramatically influence the gene selection task. Towards this end, a useful mathematical formalization was introduced with the ($(\alpha, \beta) - k$−FEATURE SET problem). This has made it possible to find genetic signatures for Alzheimer's disease [33], the molecular classification of cancer [36], and even the prediction of US presidential election results [37].

We will see that the $(\alpha, \beta) - k-$FEATURE SET Problem can be formalized as a problem of finding a certain type of subgraph in a bipartite graph. In addition, such a subgraph contains a biclique $K_{k',k}$ where k' is the minimum of the values α and β.

74.4.1 The $(\alpha, \beta) - k-$Feature Set Problem

We use the $(\alpha, \beta) - k-$FEATURE SET Problem as our mathematical formalization of the problem of interest since we aim to obtain robust genetic signatures of the different types of cancer. Robustness is obtained via some redundance in the genes/features that allow the discrimination. As a consequence, our genetic signatures guarantee that, if a feasible solution exists for the dataset of interest, at least α genes will help discriminate between any two samples of different classes. In addition, the genetic signature will have at least β genes with similar values between any two samples of the same class.

The $(\alpha, \beta) - k-$FEATURE SET Problem is a generalization of the $k-$FEATURE SET [38]. Thus it is trivially $\mathcal{NP}$-complete, because the $k-$FEATURE SET is $\mathcal{NP}$-complete [39] (a $k-$FEATURE SET corresponds to an $\alpha = 1, \beta = 0$ $(\alpha, \beta) - k-$FEATURE SET). Formally, $(\alpha, \beta) - k-$FEATURE SET

- *Instance.* A set of m examples $X = \{x^{(1)}, \ldots, x^{(m)}\}$, such that for all i, $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}, t^{(i)}\} \in \{0, 1\}^{n+1}$, and three integers $k > 0$, and $\alpha, \beta \geq 0$.
- *Question.* Does there exist an $(\alpha, \beta) - k$-feature set S, $S \subseteq \{1, \ldots, n\}$, with $|S| \leq k$ and such that:
  - for all pairs of examples $i \neq j$, if $t^{(i)} \neq t^{(j)}$ there exists $S' = S'(i, j) \subseteq S$ such that $|S'| \geq \alpha$ and for all $l \in S'$, $x_l^{(i)} \neq x_l^{(j)}$?
  - for all pairs of examples $i \neq j$, if $t^{(i)} = t^{(j)}$ there exists $S' \subseteq S$ such that $|S'| \geq \beta$ and for all $l \in S'$, $x_l^{(i)} = x_l^{(j)}$?

where the set S' is not necessary the same for all pairs of examples, so we have written $S' = S'(i, j)$.

74.4.2 Parameterized Intractability

The $\mathcal{NP}$-completeness of k-FEATURE SET implies that there may exist no polynomial-time algorithm for this problem. A natural parameter to consider is the cardinality of the feature subset.

Theorem 74.2 (Cotta and Moscato [40])

Unless $FPT = W[2]$, the $(\alpha, \beta) - k-$FEATURE SET *problem is not FPT for parameter k.*

Thus, unlike as was the case for CLIQUE, we cannot rely on an FPT algorithm. So heuristic algorithms are a reasonable alternative for this problem. Even if the problem is not FPT, however, it may permit powerful reduction rules that can shrink problem size. The application of such rules may in extreme cases even turn large instances of $\mathcal{NP}$-hard problems into small instances solvable by hand or enumeration [41]. A greedy heuristic coupled with reduction rules for this purpose is very useful to address this problem.

74.4.3 Reduction Rules for the $(\alpha, \beta) - k-$Feature Set Problem

We will explain these rules with the help of the RED-BLUE DOMINATING SET problem and consider the case of the $(1, 0) - k-$FEATURE SET problem first. If I is an instance of this problem we first transform it to an instance of the RED-BLUE DOMINATING SET using the following procedure: a bipartite graph $G(V_1 \cup V_2, E)$ is constructed such that:

- There is a red vertex $g_i \in V_2$ for each feature/gene in I, that is, $|V_2| = n$.
- There is a blue vertex $p_{jk} \in V_1$ for each pair of examples $x^{(j)}$ and $x^{(k)}$ such that $t^{(j)} \neq t^{(k)}$.
- There is an edge (g_i, p_{jk}) whenever $x_i^{(j)} \neq x_i^{(k)}$.

We leave to the reader the task of verifying that I is a yes-instance if, and only if, there exists a red dominating set $D \subseteq V_2$ such that $|D| \leq k$ and it can be generalized to the $(\alpha, 0) - k-$FEATURE SET (requesting that D be $\alpha-$dominating, i.e., that at least α vertices in D dominate each vertex in V_1 [42]). The final generalization to the $(\alpha, \beta) - k-$FEATURE SET problem is easy from here: A tripartite graph

$G(V_1 \cup V_2 \cup V_3, E)$ is constructed such that V_1, V_2, and the edges among vertices in them are as described before, and

- There is a blue vertex $c_{jk} \in V_3$ for each pair of examples $x^{(j)}$ and $x^{(k)}$ such that $t^{(j)} = t^{(k)}$.
- There is an edge (g_i, c_{jk}) whenever $x_i^{(j)} = x_i^{(k)}$.

Then an instance I would be a yes-instance if, and only if, $D \subseteq V_2$ α-dominates V_1, β-dominates V_3, and $|D| \leq k$.

We now introduce an auxiliary integer variable r_v with each vertex $v \in V_1 \cup V_3$; such that, initially, $r_p = \alpha$ for each $p \in V_1$, and $r_c = \beta$ for each $c \in V_3$; let $G(v) = \{g \in V_2 \mid (g, v) \in E\}$ be the set of vertices (genes) dominating vertex $v \in V_1 \cup V_3$; conversely, let $N(g) = \{v \in V_1 \cup V_3 \mid (g, v) \in E\}$ be the vertices in $V_1 \cup V_3$ dominated by gene $g \in V_2$. The three basic rules for this problem can be then applied as following:

R1. For each $v \in V_1 \cup V_3$ such that $r_v = |G(v)|$ do
   i. For each $g \in G(v)$, mark g as belonging to the solution.
   ii. Delete v from G.
R2. For each $v \in V_1 \cup V_3$ such that $r_v \leq 0$ delete v from G.
R3. For each $v_1, v_2 \in V_1 \cup V_3$, $v_1 \neq v_2$ such that $r_{v_1} \geq r_{v_2}$ and $G(v_1) \subseteq G(v_2)$, delete v_2 from G.

If a gene is marked, or a vertex is deleted, the following actions are taken:

Gene marking [g]: For each $v \in N(g)$ do
   i. $r_v \leftarrow r_v - 1$.
   ii. $G(v) \leftarrow G(v) \setminus \{g\}$.

Vertex deleting [v]: For each $g \in G(v)$ do $N(g) \leftarrow N(g) \setminus \{v\}$

These three rules greatly simplify the original instance by marking genes that are bound to appear in the final solution, and removing *subsumed* vertices, that is, vertices that will be dominated for sure upon domination of another vertex. The application of these rules is interleaved until the the graph cannot be further simplified.

74.4.4 Discretization of Numeric Values

In data mining, an important problem is to determine, given numeric value information, a reasonable discretization. We note that the $(\alpha, \beta) - k$—Feature Set Problem was defined as having a Boolean input matrix. This said, we need to find, for each gene a threshold value that dicotomizes the expression. For this study, we have used two different methods, one proposed by Fayyad and Irani [43] and another in which an evolutionary search strategy is applied to find a large biclique and employs the reduction rules described above [38]. The methods give similar, but different results, and we currently use them as complementary approaches to retrieve many relevant genes [33].

74.5 A Hamiltonian Path-Motivated Approach for Gene Ordering

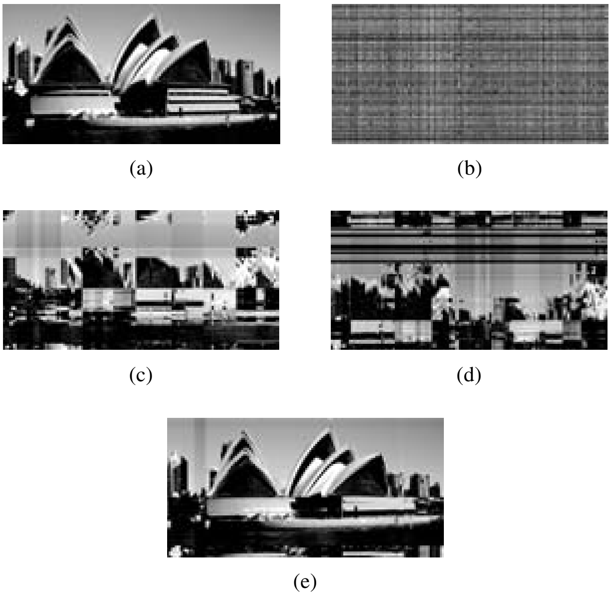
A number of approaches for the ordering of gene expression patterns have been based on combinatorial optimization. Much of the time the number of genes to be ordered can be quite large (several thousands). Researchers have therefore resorted to heuristics because finding a Hamiltonian path of minimum weight is $\mathcal{NP}$-hard. A number of heuristic and metaheuristic algorithms have been developed, with *Self Organizing Maps* (SOMs) as possibly one of the most widely used. Implementations of SOMs have found their way into commercial packages for microarray data analysis. Some software packages, both commercial and in the public domain, use some form of hierarchical clustering and *ad hoc* heuristics for the ordering of the leaves of the dendogram that represents the final clustering. Under some special but still quite practical conditions, an optimal arrangement can be found in polynomial time [44,45].

It has recently been recognized, however, that this type of approach may not always lead to results that entirely satisfy life science researchers. Gene members of the same functional group can be scattered in such orderings. An alternate objective function was introduced in Ref. [46]. If hierarchical clustering is not given as extra constraint, this leads to a problem that is $\mathcal{NP}$-hard because it contains the minimum-weight Hamiltonian path problem as a special case. The input is an integer matrix of gene expression values $G = g_{ij}$, $1 \leq i \leq n$, $1 \leq j \leq m$, where n the number of genes, m the number of samples, and g_{ij} the level of activity of gene i under condition j. We are also given a function that allows us, given any two patterns, to compute the degree of dissimilarity between them. We need to find a permutation of the genes' names $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, such that the genes with the most similar expression patterns are close to each other in the sought permutation. Now, the task is to find the permutation that minimizes the following objective function:

$$TotalCost(\pi) = \sum_{l=1}^{n} \sum_{i=\max(l-w_s,1)}^{\min(l+w_s,n)} (w_s - |l-i| + 1) D[\pi_l, \pi_i] \tag{74.1}$$

where the window size is $2w_s + 1$ (the number of genes involved in each partial distance calculation) and $D[\pi_l, \pi_i]$ represents the measure of dissimilarity between π_l and π_i. For our purpose in this chapter, we fix the parameter w_s at $\lfloor 0.01n \rfloor$ (see Ref. [46] for the influence of this parameter in the final solution). This objective function was also recently adopted for the visualization and analysis of metabolic pathways [47].

During the last decade, several combinatorial optimization problems for finding an optimal permutation have been addressed with memetic algorithms [48–50]. We also use this metaheuristic to address this problem. In addition, memetic algorithms have been introduced with the motivation of obtaining an almost linear speedup when parallelized [51] due to its inherent asychronism and low interprocessor communication requirements. In Ref. [52], it has been shown that this algorithm is very robust to individual noise measurements and was used to order genetic signatures of Alzheimer's disease [33]. They have also been applied to cancer's genetic signatures [36]. The next section shows an illustrative example (Figure 74.2) of its performance and a comparison with some of the most used methods available on the public domain.


(a)

(b)

(c)

(d)

(e)

FIGURE 74.2 Opera House-based images. (a) The original image, containing 489 rows and 971 columns; (b) randomized image, a random permutation of the image's rows and columns used to illustrate the performance of the different algorithms; (c) EBI solution, the solution from the European Bioinformatics Institute's *Expression Profiler*; (d) Eisen solution, the solution from Eisen's hierarchical clustering [53]; (e) our memetic algorithm solution.

74.6 Computational Experiments and Results

We present results on the application of these three techniques using a microarray dataset of a number of cell lines originating from different cancers. To ensure the reproducibility of our techniques, we have chosen to work with cell lines and a public domain dataset called NCI60. The original dataset and a clustering analysis was introduced in Ref. [54]. In addition, we will show how a memetic algorithm, using a similarity measure between pairs of genes (or pairs of samples), is able to obtain permutations of the rows and columns such that the final layout is highly correlated and highlights the major common groups.

In Figure 74.2, we present the results of three different algorithms for ordering microarray data. We have used an image to illustrate their main characteristics and the memetic algorithm is later used to order the genetic signatures of Figure 74.3 and Figure 74.4. Figure 74.2(a) shows the original image that contains 489 rows and 971 columns of grey-scale pixel values. The rows and columns are randomly permuted to obtain Figure 74.2(b), illustrating the task we have on real data. We present results of two of the best algorithms for analyzing microarray datasets that are available on the public domain. Figure 74.2(c), shows the results of a hierarchical clustering algorithm *European Bioinformatics Initiative* (EBI) as part of the *Expression Profiler* software tool.[2] Figure 74.2(d), proposed by Eisen et al. [53], a hierarchical clustering algorithm that also performs the ordering of the genes.[3] Finally, Figure 74.2(e) shows the result of our memetic algorithm [46], and in the three cases we have used the same algorithms to order both the rows and columns. In the rest of the chapter, we will only use the memetic algorithm to order the genetic signatures shown in all the other figures.

The original NCI60 dataset has 64 samples from 60 cell lines (i.e., two cell lines have three samples each in the set). A total of 9,703 human cDNAs have been spotted on glass microscope slides; the cDNAs thus included around 8,000 different genes. We have worked with the dataset that corresponds to Figure 2 given in Ref. [54], which helps to illustrate the power of our combinatorial approach. Again, for the purpose of illustration of the technique, we have selected only a subset of the samples that corresponds to four types of cancer: melanoma (SK-MEL-5, M-14, SK-MEL-28, UACC-257, MALME-3M, UACC-62, SK-MEL-2A), leukemia (RPMI-8226, K562, K562, K562, HL-60, MOLT-4, CCRF-CEM, SR), Colon (HCT-116, SW-620, HCT-15, KM12, HCC-2998, COLO205, HT-29), and renal (A498, RXF-393, a786-0, CAKI-1, ACHN, UO-31, TK-10). This means that we have excluded for the purpose of this study cell lines LOXIMVI (Melanoma), as well as SN12C and SNB-75 (both renal). The reason is that they seem to have, overall, a very different gene expression pattern than the others from the same class. While the reason of removing for consideration was only done to help illustrate better the power of the basic technique (providing very distinctive genetic signatures), other issues should be considered. For instance, have these cell lines remained with molecular characteristic of their parent tumors? Again, for the purpose of the illustration case, we would not include them in this study.

The first question that we would like to address could be informally phrased as: *Which are the genes that are a genetic signature of colon cancer?* An analogous question can be asked for the three other different types. We realize that this basic question is implicit in the analysis of Ref. [54] and is also implicit in several other analyses. In Ref. [54], an attempt has been made to identify “clusters” of genes that are related to a given type of cancer. Unfortunately, the authors only used the information given by the clustering algorithm. This has led them to identify genetic signatures containing only the highly expressed genes.

Figure 74.3 shows the genetic signatures of the renal, melanoma, colon, and leukemia [54] cell lines listed above. Figures 74.2(a–d), correspond to an different $(\alpha, \beta) - k$-feature sets obtained. All these genetic signatures have been obtained using a methodology first employed in Ref. [36]. Initially, an $(\alpha_{max}, \beta = 0) - k$-feature set is obtained, where α_{max} is the maximum obtainable discrimination that can be guaranteed for all pairs of samples. This means that there exists at least a pair of samples that belong to

[2] http://ep.ebi.ac.uk/EP/EPCLUST/

[3] http://rana.lbl.gov/EisenSoftware.htm

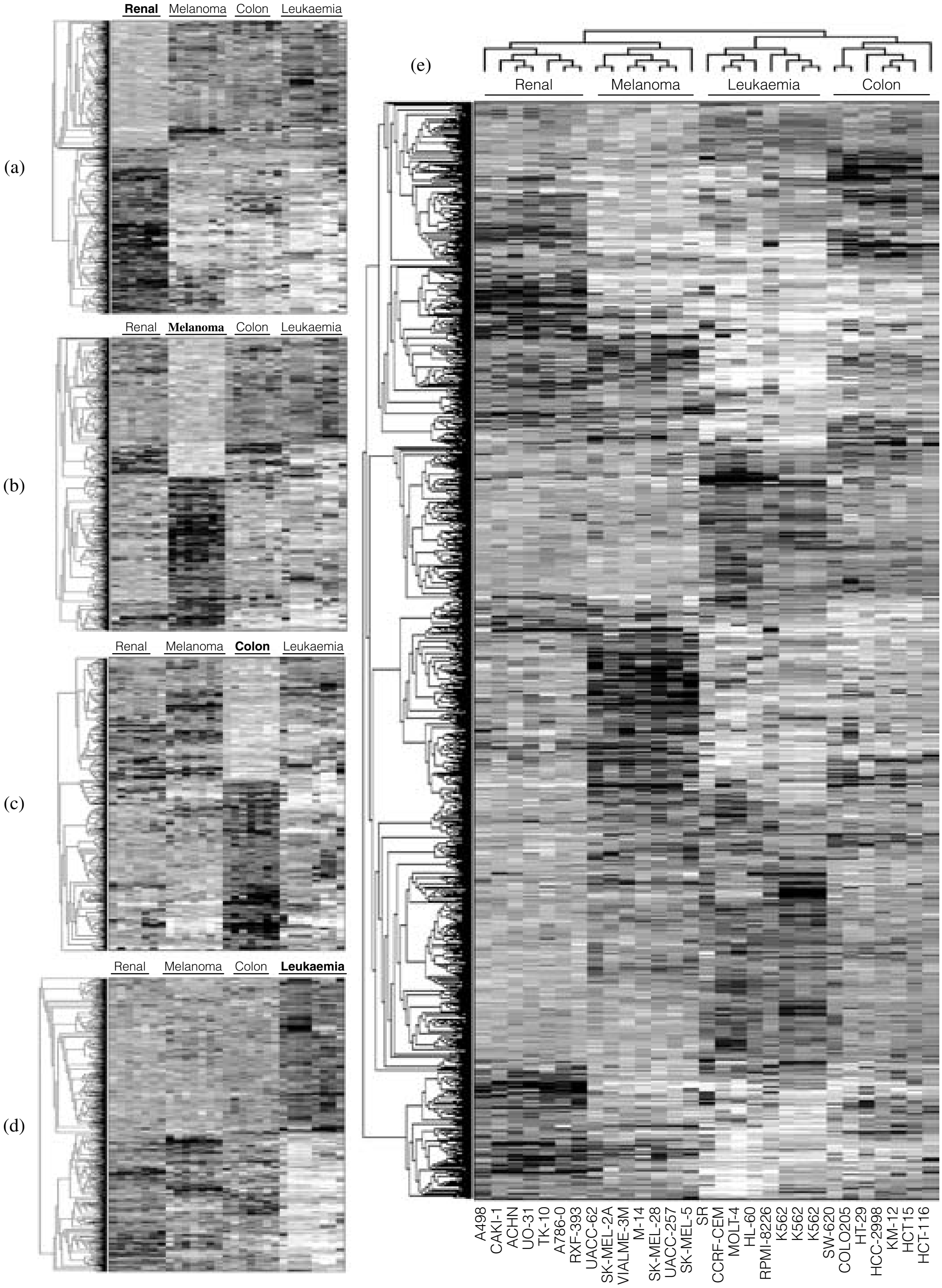


FIGURE 74.3 Signatures of the four types of cancer: (a) renal, (b) melanoma, (c) colon, and (d) leukemia. The image on the right-hand side (e) is the union of the four sets on the left-hand side and contains the profiles of 2,998 genes in 29 cell lines.

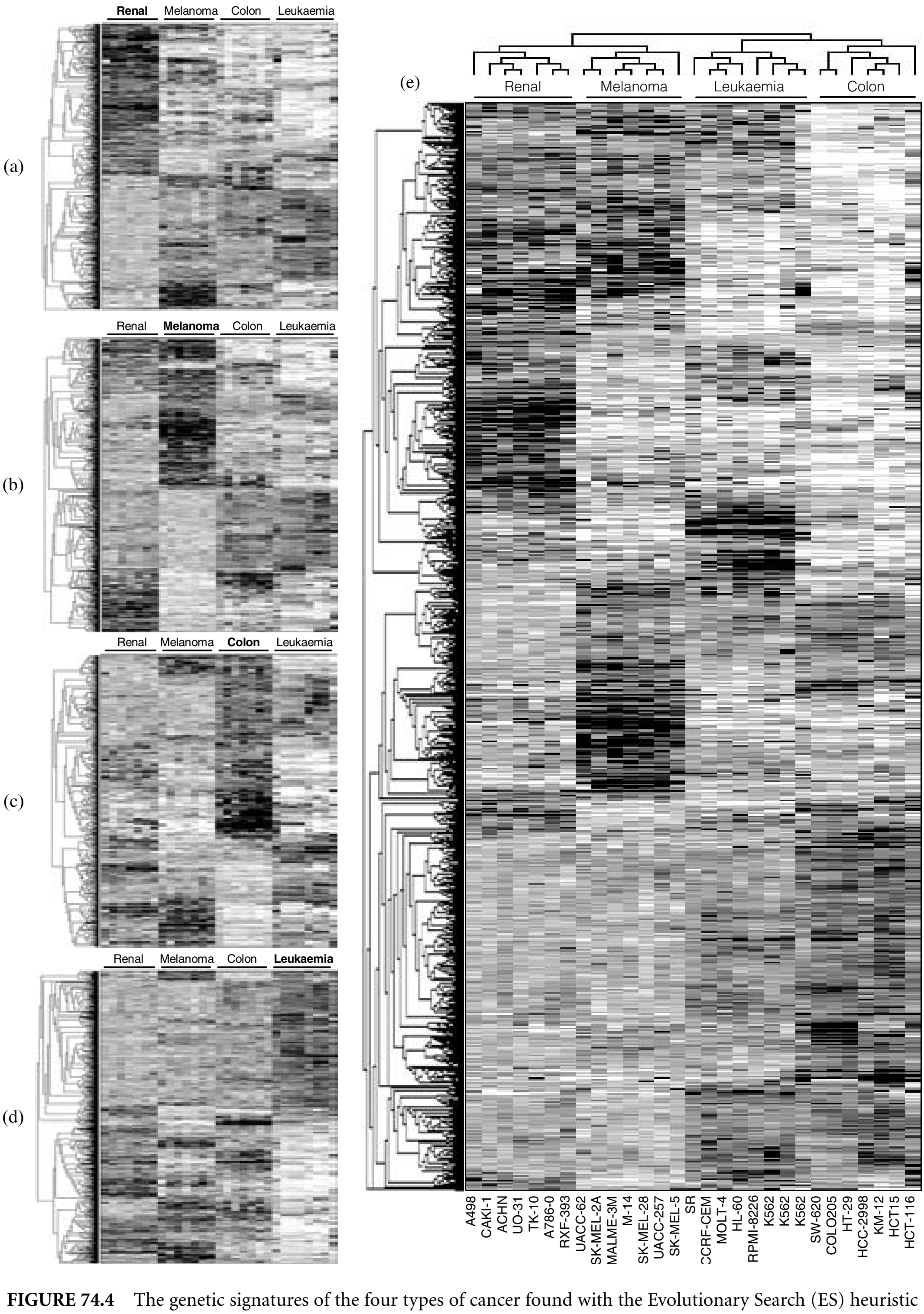


FIGURE 74.4 The genetic signatures of the four types of cancer found with the Evolutionary Search (ES) heuristic introduced in Ref. [38]. They discriminate renal (a), melanoma (b), colon (c), and leukemia (d). Their union (2,259 genes) is shown in (e). The signatures have 1,120, 1,035, 556, and 1,255 genes, respectively. The ES has an advantage for particular values of (α, β), where exact searches are too time consuming. In this case it shows comparatively similar results to the exact algorithm used for Figure 74.3.

different classes (renal vs. nonrenal) such that we can only find α_{max} differentially expressed genes. For the renal vs. nonrenal case, we have found $\alpha_{max} = 768$. The parameter β is set to zero, thus not considering the within-class similarity. We have then found a $(768, \beta = 0) - k$-feature set with the objective of minimizing the number of genes in the signature (k). We found it requires only 1,073 genes. We then proceed trying to increase the within-class similarity of our genetic signatures without incrementing the number of genes. We stop when we obtain a maximum value of β such that if we increase it by at least one unit, we cannot obtain a genetic signature with the optimal value of 1,073 (obtained when we aimed to find the minimum cardinality $(768, \beta = 0) - k$-feature set). Figure 74.3(a) shows the result: a genetic signature for renal cancer (relative to the other three types), which corresponds to a $(\alpha_{max} = 768, \beta = 655)$, $k_{opt} = 1, 073)$ feature set (where the genes are the features in this case). Analogously, Figures 74.3(b–d) correspond to the genetic signatures of melanoma $(\alpha_{max} = 714, \beta = 673, k_{opt} = 985)$, colon $(\alpha_{max} = 358, \beta = 277, k_{opt} = 521)$, and leukemia $(\alpha_{max} = 814, \beta = 743, k_{opt} = 1, 253)$, respectively.

In total, the union of the four genetic signatures has 3,832 genes, but only 2,998 are different. Figure 74.3(e) finally displays those 2,998 genes. In all cases, the order of the genes was found with the memetic algorithm allowing to identify different groups of up and down regulated genes. When the union of the four signatures is displayed as a whole, the within-class differences of the different tumors start to become evident. A clear example is given by leukemia's cell lines RPMI-8226 and SR, colon's HCT-116, and melanoma's SK-MEL-5.

74.7 Conclusions

We have shown how a combinatorial optimization approach for the problem of pattern recognition in microarray data helps to provide useful solutions to classify hundreds of genes involved in a disease. These approaches are complementary to statistical methodologies which, in turn, can benefit from the extraordinary performance of these methods to organize the data and extract interesting hypothesis for further testing and validation.

We have used publicly available data, to ensure reproducibility and for illustrative purposes. We have selected the NCI60 dataset, since it has been available since 2000 and some researchers have regarded it as "uninformative" in the past. Our results seem to indicate that this label may be related to the inadequacy of previous methodologies rather than something intrinsic to this dataset. We have shown how a combination of powerful metaheuristics and exact algorithms allow to find genetic signatures for some of the major cancer groups in the dataset.

If the genetic signatures that we have found correspond to characteristic of the tumor types in vivo, they may have several uses. At the very least, they can help in determining the true origin of a metastases without obvious primary. Possibly, the most important role of this type of analysis is to provide a molecular classification of cancer, which is novel and independent from traditional clinical taxonomies. Finally, if this classification correlated well with the characteristics in vivo, they may have a central role in personalized medicine. It could then be possible to link patients with the most appropriate tumor chemotherapy, a dreamed scenario which may be closer than we imagine.

Acknowledgments

We thank Alexandre Mendes, Pritha Mahata, and Wagner Costa for their assistance and discussion during the preparation of this chapter. We also thank Rodney Scott, John Forbes, and Alan Coates for their helpful comments.

References

[1] Gershon, D., Microarray technology—an array of opportunities, *Nature*, 416(6883), 885, 2002.

[2] Kothapalli, R., Yoder, S., Mane, S., and Loughran, T., Microarray results: how accurate are they? *BMC Bioinformatics*, 3(22), 1, 2002.

[3] Wu, L. F., Hughes, T. R., Davierwala, A. P., Robinson, M. D., Stoughton, R., and Altschuler, S. J., Large-scale prediction of saccharomyces cerevisiae gene function using overlapping transcriptional clusters, *Nat. Genet.*, 31(3), 255, 2002.

[4] Alon, U., Biological networks: the tinkerer as an engineer, *Science*, 301, 1866, 2003.

[5] Barabási, A.-L. and Oltvai, Z. N., Network biology: understanding the cell's functional organization, *Nat. Rev. Genet.*, 5, 101, 2004.

[6] Oltvai, Z. N. and Barabási, A.-L., Systems biology. Life's complexity pyramid, *Science*, 298, 763, 2002.

[7] Heymans, M. and Singh, A. K., Deriving phylogenetic trees from the similarity analysis of metabolic pathways, *ISMB (Supplement of Bioinformatics)*, 19(Suppl 1), 138–146, 2003.

[8] Bomze, I., Budinich, M., Pardalos, P., and Pelillo, M., The maximum clique problem, in *Handbook of Combinatorial Optimization*, Du, D.-Z., and Pardalos, P. M., Eds., Kluwer, Dordrecht, Suppl. Vol. A, 1–74, 1999.

[9] Bellaachia, A., Portnoy, D., Chen, Y., and Elkahloun, A. G., E-CAST: a data mining algorithm for gene expression data, *Proc. Workshop on Data Mining in Bioinformatics*, 2002, p. 49.

[10] Ben-Dor, A., Bruhn, L., Friedman, N., Nachman, I., Schummer, M., and Yakhini, Z., Tissue classification with gene expression profiles, *J. Comput. Biol.*, 7(3–4), 559–583, 2000.

[11] Ben-Dor, A., Shamir, R., and Yakhini, Z., Clustering gene expression patterns, *J. Comput. Biol.*, 6(3/4), 281, 1999.

[12] Hansen, P. and Jaumard, B., Cluster analysis and mathematical programming, *Math. Prog.*, 79(1–3), 191, 1997.

[13] Hartuv, E., Schmitt, A., Lange, J., Meier-Ewert, S., Lehrachs, H., and Shamir, R., An algorithm for clustering cDNAs for gene expression analysis, *Proc. RECOMB*, 1999, p. 188.

[14] Alter, O., Brown, P. O., and Botstein, D., Singular value decomposition for genome-wide expression data processing and modeling, *Proc. National Academy of Sciences*, Vol. 97, 2000, p. 10101.

[15] Girolami, M. and Breitling, R., Biologically valid linear factor models of gene expression, *Bioinformatics*, 20, 3021–3033, 2004.

[16] Fellows M. R. and Langston, M. A., Nonconstructive tools for proving polynomial-time decidability, *JACM*, 35, 727, 1988.

[17] Fellows, M. R. and Langston, M. A., On search, decision and the efficiency of polynomial-time algorithms, *JCSS*, 49, 769, 1994.

[18] Downey, R. G. and Fellows, M. R., *Parameterized Complexity*, Springer, Berlin, 1999.

[19] Buss, J. F. and Goldsmith, J., Nondeterminism within $\mathcal{P}$, *SIAM J. Comput.*, 22, 560, 1993.

[20] Khuller, S., The vertex cover problem, *ACM SIGACT News*, 33, 31, 2002.

[21] Nemhauser, G. L. and Trotter, L. E., Vertex packings: structural properties and algorithms, *Math. Prog.*, 8, 232, 1975.

[22] Abu-Khzam, F. N., Collins, R. L., Fellows, M. R., Langston, M. A., Suters, W. H., and Symons, C. T., Kernelization algorithms for the vertex cover problem: theory and experiments, *Proc. Workshop on Algorithm Engineering and Experiments*, New Orleans, LA, USA, 2004.

[23] Abu-Khzam, F. N., Langston, M. A., and Shanbhag, P., Scalable parallel algorithms for difficult combinatorial problems: a case study in optimization, *Proc. Int. Conf. on Parallel and Dist. Comput. and Sys.*, 2003, p. 563.

[24] Baldwin, N. E., Collins, R. L., Langston, M. A., Leuze, M. R., Symons, C. T., and Voy, B. H., High performance computational tools for motif discovery, *Proc. IEEE Int. Workshop on High Performance Computational Biology*, Santa Fe, New Mexico, USA, 2004.

[25] Abu-Khzam, F. N., Langston, M. A., Shanbhag, P., and Symons, C. T., Scalable Parallel Algorithms for FPT Problems, Technical report UT-CS-04-524, Department of CS, University of Tennessee, 2004.

[26] Dorai, M., Bouldin, D. W., Langston, M. A., and Peterson, G. D., FPGA-based solutions for the branching phase of fixed-parameter tractable computations, Manuscript, 2004.

[27] Zhang, Y., Abu-Khzam, F. N., Baldwin, N. E., Chesler, E. J., Langston, M. A., and Samatova, N. F., Genome-scale computational approaches to memory-intensive applications in systems biology, *Proc. Supercomputing Conf.*, Seattle, WA, USA, 2005.

[28] J. M. Kim, J. M., Sohn, H. Y., Yoon, S. Y., Yang, J. O., Kim, J. H., Song, K. S., Rho, S. M., Yoo, H. S., Kim, Y. S., Kim, J. G., and Kim, N. S., Identification of gastric cancer-related genes using a cDNA microarray containing novel expressed sequence tags expressed in gastric cancer cells, *Clin. Cancer Res.*, 11, 473, 2005.

[29] Hor, S., Pirzer, H., Dumoutier, L., Bauerand, F., Wittmann, S., Sticht, H., Renauld, J. C., de Waal Malefyt, R., and Fickenscher, H., The T-cell lymphokine interleukin-26 targets epithelial cells through the interleukin-20 receptor 1 and interleukin-10 receptor 2 chains, *J. Biol. Chem.*, 279(32), 3343, 2004.

[30] van't Veer, L. J., Dai, H. Y., van de Vijver, M. J., He, Y. D. D., Hart, A. A. M., Mao, M., Peterse, H. L., van der Kooy, K., Marton, M. J., Witteveen, A. T., Schreiber, G. J., Kerkhoven, R. M., Roberts, C., Linsley, P. S., Bernards, R., and Friend, S. H., Gene expression profiling predicts clinical outcome of breast cancer, *Nature*, 415(6871), 530, 2002.

[31] Spellman, P. T., Sherlock, G., Zhang, M. Q., Iyer, V. R., Anders, K., Eisen, M. B., Brown, P. O., Botstein, D., and Futcher, B., Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization, *Mol. Biol. Cell*, 9, 3273, 1998.

[32] Elkon, R., Linhart, C., Sharan, R., Shamir, R., and Shiloh, Y., Genome-wide in silico identification of transcriptional regulators controlling the cell cycle in human cells, *Genome Res.*, 13(5), 773, 2003.

[33] Moscato, P., Berretta, R., Hourani, M., Mendes, A., and Cotta, C., Genes related with Alzheimer's disease: a comparison of evolutionary search, statistical and integer programming approaches, *Proc. Evo Workshops*, Lecture Notes in Computer Science, Vol. 3449, Springer, Berlin, 2005, p. 84.

[34] Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D., and Lander, E. S., Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, *Science*, 286, 531, 1999.

[35] Pomeroy, S. L., Tamayo, P., Gaasenbeek, M., Sturla, L. M., Angelo, M., McLaughlin, M. E., Kim, J. Y. H., Goumnerova, L. C., Black, P. M., Lau, C., Allen, J. C., Zagzag, D., Olson, J. M., Curran, T., Wetmore, C., Biegel, J. A., Poggio, T., Mukherjee, S., Rifkin, R., Califano, A., Stolovitzky, G., Louis, D. N., Mesirov, J. P., Lander, E. S., and Golub, T. R., Prediction of central nervous system embryonal tumour outcome based on gene expression, *Nature*, 415(6870), 436, 2002.

[36] Berretta, R., Mendes, A., and Moscato, P., Integer programming models and algorithms for molecular classification of cancer from microarray data, *Proc. Australasian Computer Science Conf.,* CRPIT, Vol. 38, 2005, p. 361.

[37] Moscato, P., Mathieson, L., Mendes, A., and Berretta, R., The electronic primaries: predicting the U. S. presidency using feature selection with safe data reduction, *Proc. Australasian Computer Science Conference,* CRPIT, Vol. 38, 2005, p. 371.

[38] Cotta, C., Sloper, C., and Moscato, P., Evolutionary search of thresholds for robust feature set selection: Application to the analysis of microarray data, *Proc. Evo Workshops*, Lecture Notes in Computer Science, Vol. 3005, Springer, Berlin, 2004, p. 21.

[39] Davies, S. and Russell, S., *NP*-completeness of searches for smallest possible feature sets, *Proc. AAAI Symp. on Intelligent Relevance*, 1994, p. 41.

[40] Cotta, C. and Moscato, P., The k-FEATURE SET problem is $W[2]$-complete, *JCSS*, 67(4), 686, 2003.

[41] Weihe, K., Covering trains by stations or the power of data reduction, *Proc. Algorithms and Experiments*, 1998, p. 1.

[42] Harant, J., Pruchnewski, A., and Voigt, M., On dominating sets and independent sets of graphs, *Comb. Prob. Comput.*, 8, 547, 1999.

[43] Fayyad, U. M. and Irani, K. B., Multi-interval discretization of continuous-valued attributes for classification learning, *Proc. IJCAI*, 1993, p. 1022.

[44] Biedl, T., Brejova, B., Demaine, E. D., Hamel, A. M., and Vinar, T., Optimal Arrangement of Leaves in the Tree Representing Hierarchical Clustering of Gene Expression Data, Technical Report 2001-14, Univ. of Waterloo, Canada, 2001.

[45] Bar-Joseph, Z., Demaine, E. D., Gifford, D. K., Hamel, A. M., Jaakkola, T. S., and Srebro, N., K-ary Clustering with optimal leaf ordering for gene expression data, *Proc. Int. Workshop Algorithms in Bioinformatics*, Lecture Notes in Computer Science, Vol. 2452, Springer, Berlin, 2002, p. 506.

[46] Cotta, C., Mendes, A., Garcia, V., França, P., and Moscato, P., Applying memetic algorithms to the analysis of microarray data, *Proc. European Workshop on Evolutionary Bioinformatics*, Lecture Notes in Computer Science, Vol. 2611, Springer, Berlin, 2003, p. 22.

[47] Conrad, T., New approaches for visualizing and analysing metabolic pathways, *Proc. Australian Undergraduate Students' Computing Conf.*, 2004, p. 48.

[48] Moscato, P., An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of tabu search, *Ann. Oper. Res.*, 41, 85, 1993.

[49] Merz, P. and Freisleben, B., Memetic algorithms for the traveling salesman problem, *Complex Syst.*, 13(4), 297, 2001.

[50] Buriol, L. S., França, P. M., and Moscato, P., A new memetic algorithm for the asymmetric traveling salesman problem, *J. Heuristics*, 10(5), 483, 2004.

[51] Mendes, A., Cotta, C., Garcia, V., França, P., and Moscato, P., Gene ordering in microarray data using parallel memetic algorithms, *Proc ICPP Workshops*, 2005, p. 604.

[52] Moscato, P., Berretta, R., and Mendes, A., A new memetic algorithm for ordering datasets: applications in microarray analysis, *Proc. Metaheuristics Int. Conf.*, Vienna, Austria, 2005.

[53] Eisen, M. B., Spellman, P. T., Brown, P. O., and Botstein, D., Cluster analysis and display of genome-wide expression patterns, *Proc. National Academy of Sciences of the USA 95,* 1998, p. 14863.

[54] Ross, D. T., Scherf, U., Eisen, M. B., Perou, C. M., Rees, C., Spellman, P., Iyer, V., Jeffrey, S. S., Van de Rijn, M., Waltham, M., Pergamenschikov, A., Lee, J. C.F., Lashkari, D., Shalon, D., Myers, T. G., Weinstein, J. N., Botstein, D., and Brown, P. O., Systematic variation in gene expression patterns in human cancer cell lines, *Nat. Genet.*, 24, 227, 2000.

75

Approximation Algorithms for the Primer Selection, Planted Motif Search, and Related Problems

Sanguthevar Rajasekaran
University of Connecticut

Jaime Davila
University of Connecticut

Sudha Balla
University of Connecticut

75.1 Primer Selection Problem

In this chapter, we consider two problems from computational biology, namely, primer selection and planted motif search (PMS). The closest string and the closest substring problems are closely related to the PMS problem. All of these problems have been proven to be NP-hard. We survey some representative approximation algorithms that have been proposed for these problems.

The problem of selecting primers for polymerase chain reaction (PCR) and multiplex PCR (MP-PCR) experiments is important in computational biology and has drawn the attention of numerous researchers in the recent past. This is a minimization problem that seeks the minimum set of primers required for a given set of DNA sequences as the input. The primers selected for the input set could be of two different categories, namely, nondegenerate primers and degenerate primers. The latter method of designing degenerate primers for a given input set gives rise to a variant of the primer selection problem (PSP) called the degenerate primer selection problem (DPSP). These two variants have been proven to be NP-complete in the literature and also intractable to approximation within a constant to the optimal solution [1,2]. Thus, a number of heuristics have been proposed in the literature to select primers and in this chapter, we discuss them in detail. The primers can be viewed as motifs occurring in the input set and hence this problem is related to the problem of identifying motifs in deoxyribonucleic acid (DNA) sequence data.

75.1.1 Background Information

PCR is a molecular biological method for amplifying, that is, creating multiple copies of, DNA sequences. In its basic form, PCR requires a pair of synthetic DNA sequences, called forward and reverse primers, which are short single-stranded DNA strings, typically 15–20 nucleotides in length, which exactly match the beginning and end of the DNA fragment to be amplified.

Multiplex PCR (MP-PCR) is a variant of PCR, which enables simultaneous amplification of multiple DNA fragments of interest in one reaction by using a mixture of multiple primers [3]. This method has been applied in many areas of DNA testing, including analyses of deletions, mutations, and polymorphisms, and, more recently, in genotyping applications requiring simultaneous analysis of up to thousands of markers. A set of nondegenerate primers is selected on each end of the regions to be amplified for the given input set of DNA sequences. This is the basic version of the problem and is called the PSP in the literature. We discuss some of the salient algorithms proposed for this problem in some detail in this chapter.

The presence of multiple primers in MP-PCR can lead to severe problems, such as unintended amplification products caused by mispriming or lack of amplification because of primer cross-hybridization. To minimize these problems, it is critical to minimize the number of primers involved in a single MP-PCR reaction, particularly when the number of DNA sequences to be amplified is large. This can be achieved by selecting primers that would simultaneously act as forward and/or reverse primers for several of the DNA sequences in the input set. A recent technique that enables higher degrees of *primer reuse* is to allow more than one nucleotide at some of the positions of the primer. Remarkably, such primers, called degenerate primers [4], are as easy to synthesize as regular primers since their synthesis requires the same number of biochemical steps (the only difference is that one must add multiple nucleotides in some of the synthesis steps). The degeneracy of a degenerate primer is the number of distinct nondegenerate primers that could be formed out of it. For example, if the degenerate primer $p_d = \mathrm{A\{CT\}GC\{ACG\}T\{GA\}}$, it has degeneracy 12; the distinct nondegenerate primers represented in p_d are ACGCATG, ACGCATA, ACGCCTG, ACGCCTA, ACGCGTG, ACGCGTA, ATGCATG, ATGCATA, ATGCCTG, ATGCCTA, ATGCGTG, and ATGCGTA. Since highly degenerate primers may give excessive mispriming, a bound on the degeneracy of a primer is typically imposed, leading to a variant of the primer selection problem called the DPSP, discussed in detail in the following sections of this chapter.

75.1.2 Polymerase Chain Reaction

The PCR experiment is conducted in a series of cycles, typically 30–40 in number, each cycle divided into three major steps. It requires several components (called the reaction mixture) that are placed in tubes, called the reaction tubes, which are repeatedly heated and cooled in an automated equipment called the thermal cycler. The components that make up the reaction mixture are a DNA template or the sequence that needs to be amplified, two primers that define the start and the end of the region to be amplified, nucleotides from which the new DNA is built by the DNA-polymerase and a suitable chemical environment provided by the buffer.

The three steps that constitute each cycle of the PCR are as follows:

Step 1. The double-stranded DNA is heated to around 94–96°C to break the hydrogen bonds that connect the two DNA strands and separate them. This step is called denaturing.
Step 2. In this step, called annealing, the temperature is lowered so the primers can attach themselves to the single DNA strands. The temperature of this stage is usually 5°C below melting temperature of the primers (45–60°C).
Step 3. The DNA-polymerase fills in the missing strands in this step called elongation. It starts at the annealed primer and works its way along the DNA strand and, typically, takes place at a temperature of around 72°C.

The total time for a single cycle of the PCR is 3–5 min. Because both strands of the DNA sequence are copied during PCR, there is an exponential increase of the number of copies of the sequence. If there is one copy of the sequence before the cycles start, after one cycle, there will be two copies; after two cycles, there will be four copies; three cycles will result in eight copies; and so on.

The quality of the amplifications depends very largely on the primers used in the experiment, thus making the primer selection a very important process. The melting temperature of a primer is the temperature at which at least half of the primer binding sites are occupied in the step 1 above and increases with the increase in the length of the primer. However, very short primers, although they have low melting temperatures, would result in binding to many locations in the DNA sequence leading to mispriming. Thus, there arise some experimental constraints for the selection of PCR primers, referred to as the biological constraints:

1. The GC content (the number of G's and C's in the primer) of the primers should be around 40–60% of its length.
2. The length of the primers should be chosen in such a way that they do not bind themselves to several positions of the DNA sequence.
3. There should not be any complementarity in the primers, that is, they should not be self-complementary, for example, the primer 5′-GCGGTATACCGC-3′ is self-complementary, and they should not be complementary to one another, for example, the primers 5′-CGAAATGCCG-3′ and 5′-CGGCATTTCG-3′ are complementary to each other.
4. The melting temperatures of both primers should not differ by more than 5°C and the melting temperature of the DNA sequence should not differ from that of the primers by more than 10°C.

75.1.3 Terminology

In this section, the terminology adopted to explain the problems under discussion and the algorithms proposed for the same is given in detail.

Let $S = \{S_1, S_2, \ldots, S_n\}$ be the set of input sequences defined over the DNA alphabet $\Sigma = \{A, C, G, T\}$. Let Σ^* denote the set of all finite strings defined over the alphabet Σ. Let l_i be the length of the sequence S_i, $1 \leq i \leq n$. Let k be the length of the primer designed for the input set. The number of k-mers (a k-mer is a substring of length k) possible from each input string S_i is $(l_i - k + 1)$, $1 \leq i \leq n$. Let P be the set of all k-mers of the input set S. A primer $p \in P$ of length k is said to cover a subset S' of the set of input sequences S, iff p is a substring of every sequence in S'. The primer that is designed to bind at the 5′ end of the sequences is called a forward primer and the one designed to bind at the 3′ end is called a reverse primer. The PSP is defined as follows.

Definition 75.1 (PSP)

The PSP is to minimize the size of the subset P' of P such that the primers of P' collectively cover the input set S, and every sequence S_i, $1 \leq i \leq n$, has at least one forward *and one* reverse *primer in P'.*

An *optimal cover* for S is defined as the set P' of minimum size.

A degenerate primer p_d is a primer of length k with one or more symbols of Σ occurring in each position. The *degeneracy* d of the degenerate primer p_d is the product of the number of symbols in each position of the primer, that is, $d(p_d) = \Pi_{i=1}^{k} |p_d[i]|$.

The *degeneracy* of a degenerate primer is also the number of distinct nondegenerate primers that could be formed out of it. For example, the degenerate primer p_d = A{CT}GC{ACG}T{GA} has a degeneracy of 12; the distinct nondegenerate primers represented in p_d are ACGCATG, ACGCATA, ACGCCTG, ACGCCTA, ACGCGTG, ACGCGTA, ATGCATG, ATGCATA, ATGCCTG, ATGCCTA, ATGCGTG, and ATGCGTA. The degenerate primer p_d is said to cover an input sequence S_i iff one of the nondegenerate primers represented in p_d is a substring of S_i. If a degenerate primer of length k covers m of the given n input sequences, it is said to have a coverage of size m.

The decision version of Degenerate Primer Design Problem (known as DPD [41]) is to find if there exists a degenerate primer of length k and degeneracy at most d that has a coverage of m for a given input set of n sequences. As the length of the primer k is decided beforehand, the algorithms that have been designed for this problem try to optimize either the degeneracy d or the coverage m and hence there are two variants of the DPD problem. The former is called the *Minimum Degeneracy Degenerate Primer Design Problem (MD-DPD)* and attempts to find a degenerate primer of length k and minimum degeneracy d_{min} that covers all the n input sequences. The latter is called the *Maximum Coverage Degenerate Primer Design Problem (MC-DPD)* that identifies a primer of length k and degeneracy at most d that covers a maximum number of the given n input strings. Linhart and Shamir [41] formulated the above versions of the problem. The formulation of MC-DPD above is for identifying one degenerate primer and can be extended to find a set of degenerate primers to cover a given set of input sequences as follows.

Definition 75.2 (**DPSP**)

Given a set S of n input sequences (DNA sequences) and integers k and d, find a set of degenerate primers P_d such that each primer in P_d has a degeneracy of at most d, the set P_d covers all the input strings S, that is, every sequence S_i, $1 \leq i \leq n$, has at least one forward and one reverse primer in P_d.

In the following sections, we discuss some salient algorithms that have been proposed for PSP and DPSP. Both these variants have been proven to be NP-complete in the literature. These proofs are described next.

75.1.4 NP-Completeness of Primer Selection Problem and Degenerate Primer Selection Problem

Pearson et al. [2] formulated the problem of finding an optimal cover for the PSP version of primer selection as follows:

Optimal Primer Cover Problem (OPCP). Given an input set S of DNA sequences and integer k, find an optimal cover of S, the primer length being k.

They proved the NP-completeness of OPCP by transforming the Minimum Set Cover problem to OPCP.

Minimum Set Cover Problem (MSCP). Let $\mathcal{F} = \{F_j\}$ be a finite family of sets. Let $\mathcal{F}'$ be a subset of $\mathcal{F}$. $\mathcal{F}'$ is a cover of $\mathcal{F}$ iff

$$U = \bigcup_{F \in \mathcal{F}'} F = \bigcup_{F \in \mathcal{F}} F$$

The decision version of MSCP is, for a given family of sets $\mathcal{F}$ and an integer f, to determine if $\mathcal{F}$ has a cover $\mathcal{F}'$ such that $|\mathcal{F}'| \leq f$.

Let the number of primers in an optimal cover solution for OPCP be q. Let (S, P, q) denote an instance of OPCP and $(\mathcal{F}, f)$ denote an instance of MSCP. An arbitrary instance $(\mathcal{F}, f)$ of MSCP is transformed into an instance (S, P, q) of OPCP such that (S, P, q) has a solution iff $(\mathcal{F}, f)$ has a solution, as given below. Let $U = \cup_{F \in \mathcal{F}} F$.

Let $q = f$; and $\Sigma = \{0, 1, b_1, b_2, b_3, \ldots, b_{|U|}\}$. Here the b_i's $(1 \leq i \leq |U|)$ are unique symbols used as separators as explained next. Let $k = \log_2 |\mathcal{F}|$. Construct the set S over the alphabet Σ as follows. Every $F_j \in \mathcal{F}$ is encoded by a unique string v_j, of length k, over the alphabet $\Sigma' = \{0,1\}$, Σ' being a subset of Σ. Every $S_i \in S$ represents a unique element $u_i \in U$, S_i encoding details about the subsets $F_j \in \mathcal{F}$ in which element u_i is present. This is achieved by concatenating the string $v_j b_i$ to S_i of all $F_j \in \mathcal{F}$ in which u_i is present. Note that b_i acts as a unique string separator in S_i.

In the above transformation of the MSCP instance to the OPCP instance, the size of the alphabet Σ varies with the size of U. But for input sets, which are DNA sequences, the alphabet Σ is fixed, that is, $\Sigma = \{A, C, G, T\}$. To transform an arbitrary MSCP instance to an OPCP instance using the fixed alphabet, it is sufficient to represent $\{0,1\}$ above using $\{a,c\}$ and $\{b_1, b_2, b_3, \ldots, b_{|U|}\}$ using $\{g,t\}$. Thus, the OPCP for the DNA alphabet is NP-complete. □

Hence we get the following theorem.

Theorem 75.1

PSP is NP-complete.

Linhart and Shamir [1] prove the NP-completeness of DPSP by proving that the Minimum Primers DPD problem (MP-DPD), a special case of DPSP where every input string is of length k, is NP-complete for $|\Sigma| \geq 2$. The proof is based on a reduction from the Minimum Bin Packing problem (MBPP).

Minimum Bin Packing Problem (MBPP). Given are q positive integers or items $a_1, a_2, a_3, \ldots, a_q$, two integers b (the number of bins) and c (the capacity). The goal is to find if the q items can be packed into the b bins such that the total sum of the items in each bin is at most c.

Given an instance of MBPP, the instance for DPSP can be constructed as follows: Let $A = \sum_{i=1}^{q} a_i$; $\Sigma = \{0,1\}$; $k = A$; $d = 2^c$; and $|P_d| = b$. The set S is constructed as follows: each string $S_i \in S$ is of length A and is over the alphabet Σ. $S_i = s_i^1, s_i^2, \ldots, s_i^A$, where $s_i^j = 1$ when $A_i \leq j \leq A_i + a_i$ and $s_i^j = 0$ otherwise. Here $A_i = \sum_{x=1}^{i-1} a_x$.

The size of the set P_d is set to b and so the goal is to find if there is a P_d of size b, with primers of length A and degeneracy 2^c that cover all the q input strings. This polynomial reduction of MBPP to DPSP proves that a solution to MBPP exists iff a solution to MP-DPD exists and hence the following theorem arises.

Theorem 75.2

DPSP is NP-complete.

Given that PSP and DPSP are NP-hard, researchers have devised several approximation algorithms for these problems. Quality bounds have been proven for some of these algorithms. For the other algorithms, the quality has been measured only empirically. We describe some of the approximation algorithms that have been proposed for PSP and DPSP next.

75.1.5 Algorithms for PSP

In this section, we will survey some of the salient algorithms from the literature that have been proposed for the PSP.

Pearson et al. [2] have proposed a simple greedy algorithm for MSCP called PSP-Greedy. The output of this algorithm is guaranteed to be within an $O(\log n)$ factor of the optimal. They have also proposed an exact branch and bound algorithm, which is not discussed here.

Algorithm **PSP-Greedy** can be used to select forward and the reverse primers for a given set of DNA sequences in two separate steps, namely, by considering the first, say, r nucleotides of the sequences to select the set of forward primers and then the last r nucleotides to select the reverse primers. Another approach is to consider the first r nucleotides and the complement of the last r nucleotides of each sequence, thus building an input dataset of $2n$ sequences of length r each. The latter approach was described by Souvenir et al. [5] to design degenerate primers. Note that the value r must be chosen carefully such that $(l_i - 2r) > 0$ for $1 \leq i \leq n$ to assure that every DNA sequence in the input would have an amplified product of length strictly >0.

```
Algorithm PSP-Greedy {
        Let $P$ be the collection of primers selected; initially, $P := \emptyset$;
        Let $R$ be the set of remaining (uncovered) sequences; initially, $R := \{1, 2, \ldots, n\}$.

        Let $C$ be the collection of $k$-mers from all the $n$ input sequences; Note that each
        input sequence $S_i$, $1 \leq i \leq n$, will have $(r - k + 1)$ $k$-mers, $r$ being the
        length of $S_i$. Each element of $C$ is a tuple of the form $< k - mer, i >$, $i$ being the
        sequence to which the $k$-mer belongs to.

        Sort the collection $C$ such that its $k$-mers are in lexicographic order. Scan
        through $C$ and identify unique $k$-mers and the list of sequences in which they
        are present (denoted by the second value in each tuple), called their coverage.
        Let $L$ be the list of all such unique $k$-mers and their coverage.
        While ($R$ is not empty) do {
            Pick $p$ from $L$, $p$ being the $k$-mer that has coverage of maximum
            cardinality among all the elements of $L$;
            $L := L - \{p\}$; $P := P \cup \{p\}$;
            $R := R - \{$coverage of $p\}$;
            For each $q \in L$ do {
                $\{$coverage of $q\} := \{$coverage of $q\} - \{$coverage of $p\}$;
            }
        }
        Output $P$;
}
```

It is obvious from its description that PSP-Greedy does not consider the biological constraints seen in the earlier section to select primers. The first efforts in this direction came from Doi and Imai [6–8], who proposed another greedy heuristic, essentially a modification of PSP-Greedy that considered biological constraints such as *GC-content* and *complementarity* in selecting primers. Their algorithm also considered length constraint of the amplified product, namely, the minimum length constraint that ensures that the minimum length of the amplified products is at least of a prespecified length l_{min} (instead of 0 as described above). Algorithms in Ref. [9] also consider length constraints in designing primers for MP-PCR.

75.1.6 Algorithms for Degenerate Primer Selection Problem

This section discusses some of the known algorithms for the DPSP. Rose et al. [10] proposed algorithm COnsensus-DEgenerate Hybrid Oligonucleotide Primers (CODEHOP) that designs hybrid primers with nondegenerate consensus clamp at the $5'$ region and a degenerate $3'$ core region. In an effort to identify genes belonging to the same family, Fuchs et al. [11] devised a two-phase algorithm called DEciphering Families Of Genes (DEFOG). In its first phase, DEFOG introduces degeneracy into a set of nondegenerate primer candidates selected because of their best entropy score. Linhart and Shamir [1] proposed an algorithm called Highly DEgeNerate (HYDEN) for the first phase of DEFOG. Wei et al. [12] contributed an algorithm based on clustering called DePiCt that designs primers of low degeneracy and high coverage for a given set of aligned amino acid sequences. Souvenir et al. [5] proposed the Multiple Iterative Primer Selector (MIPS) algorithm for a variation of DPSP, discussed in their paper as the Partial Threshold Multiple Degenerate Primer Design (PT-MDPD). Algorithms HYDEN, degenerate primer design via clustering (DePiCt), and MIPS are explained in some detail in this section.

75.1.6.1 Algorithm HYDEN

The HYDEN algorithm [1] performs the first phase of DEFOG [11]. For a given set of input sequences, HYDEN is run separately on the datasets of the first r residues from every sequence to select the forward degenerate primers (say, the set P_f) and the last r residues from each sequence to select the reverse degenerate primers (say, the set P_r). Then, the desired set $P_d = P_f \cup P_r$. For a given run, HYDEN designs a degenerate primer that covers the maximum number of the sequences that are yet to be covered (initially, all sequences are yet to be covered), adds the primer to the output set, removes the sequences that it had covered, and repeats the same procedure until all sequences are covered. To select one degenerate primer for the set of sequences alive at a given time of its execution, HYDEN employs a three-phased approach described as follows.

Phase 1. Named HYDEN-Align, this phase identifies highly conserved regions of the given set of DNA sequences by locating ungapped local alignments that contribute towards a low entropy score. It enumerates all substrings of length k (k-mers) in the input set, generates alignment score for each substring by finding best matches to it with respect to its hamming distance with substrings of other strings in the input. We know that the total number of k-mers possible in the input is $O(nr)$. HYDEN-Align considers all such possibilities and obtains $O(nr)$ alignments. Therefore, the runtime of HYDEN-Align is $O(n^2r^2k)$. A subset of these alignments, determined by another input parameter (say a, (i.e.) the 'a' *best alignments*), that have a low entropy score is considered for the next phase. The authors also give a simple heuristic that will speed up this phase, namely, initially each such alignment is generated only on a subset of the input strings (say ϵn), a subset a' ($a' > a$) of these alignments that have the best entropy scores are selected; for each partial alignment selected, a full alignment is generated, the entropy scores calculated and the best a alignments are selected for the next phase based on the complete entropy scores. This reduces the runtime of this phase to $O(knr(\epsilon L + a'))$.

The entropy scores are calculated as follows. Let A be a given ungapped alignment. A consists of n k-mers one from each input sequence. Let D_A denote a column distribution matrix of A. The column distribution matrix D_A is a two-dimensional matrix of size $|\Sigma| \times k$, where $D_A[\sigma, j]$, $1 \leq \sigma \leq |\Sigma|$ and $1 \leq j \leq k$, has the value equal to the count or the number of occurrences of the symbol $\sigma (\sigma \in \Sigma)$ in column j of the alignment A. The entropy score E_A of alignment A is given as

$$E_A = -\sum_{j=1}^{k}\sum_{\sigma\in\Sigma}[(D_A[\sigma,\ j]/n) * \log_2(D_A[\sigma,\ j]/n)]$$

The lower the entropy score, the less will be the variation of symbols in the alignment A. Thus, greater are the chances of finding a k-mer that would cover many of the input sequences.

Phase 2. In this phase two procedures, namely HYDEN-Contraction and HYDEN-Expansion, are run on the set of alignments from the first phase. HYDEN-Contraction starts with a complete degenerate primer (p_c) of degeneracy 4^k, proceeds by removing the symbol from a position at which it has occurred the minimum number of times in the given alignment until the target degeneracy d is achieved. However, HYDEN-Expansion starts from a nondegenerate primer (p_e), that is, the k-mer from which the alignment was obtained, adds to it symbols one at a time at positions where the symbol added has occurred the maximum number of times in the alignment, increasing the degeneracy until the target degeneracy is achieved. Both the procedures use D_A of each alignment A to eliminate and add symbols in the primers p_c and p_e, respectively. Two such primers are designed for every alignment in the set, and a subset of these primers that have maximum coverage is considered for the third phase (the size of the subset is given as an input parameter, say b, that is, the b best primers of the $2a$ primers designed). Each run of HYDEN-Contraction or HYDEN-Expansion takes $O(knr)$ time and there are 'a' alignments for which the primers are designed, thus the runtime of phase 2 is $O(aknr)$.

Phase 3. In this phase, called the HYDEN-Greedy, attempt is made to improve the primers selected from Phase 2 using a greedy hill-climbing approach, trying to exclude symbols in some positions of a given primer and include symbols in other positions to increase coverage.

Although there is no theoretical guarantee on the performance of algorithm HYDEN, the authors have reported good practical performance in experiments on real biological data.

75.1.6.2 Algorithm DePiCt

This algorithm proposed by Wei et al. [12] designs degenerate primers of low degeneracy and high coverage from a given multiple alignment of amino acid (or protein) sequences. It adopts clustering techniques to group the set of input sequences, thus ensuring that sequences that belong to a given cluster would have regions significantly conserved in them, which would enable the design of a pair of degenerate primers for them. Conserved regions of a cluster are determined using a novel scoring technique called the BlockSimilarity scoring. The degenerate primers are then obtained by reverse translation of the amino acids in the conserved regions to corresponding nucleotides.

The *Genetic Code* consists of triplets of nucleotides, called codons, each such codon encoding one of the 20 amino acids that are used in the synthesis of proteins in organisms. As the DNA alphabet has 4 symbols, there are 64 triplets in the genetic code, leading to some redundancy that many of the amino acids are encoded by more than one codon. For example, the amino acid proline (P) is encoded by four codons, namely, CCT, CCG, CCC, and CCA. It is obvious to see that many amino acids have very similar codons too, although they may be very different in their physical and chemical properties. For example, another amino acid alanine (A) is encoded by the codons GCT, GCG, GCC, and GCA. In calculating the similarity between the input sequences algorithm DePiCt considers proline and alanine in our example to be *similar* as they differ only by one nucleotide in the first position of the codons that encode them.

DePiCt adopts *hierarchical clustering* to cluster the set of input sequences into groups that have conserved regions. Initially, there are n groups, each group consisting of one input sequence. A series of iterations are performed to regroup the sequences in the groups based on their similarities. In each iteration, sequences of two groups that have the highest similarity score are grouped together into one, if the resultant group is a *valid* cluster. A *valid* cluster is one that has at least one conserved block of length greater than or equal to the minimum required product length or two blocks separated by a length in the range of the minimum required product length and the maximum required product length. These minimum and maximum product lengths are specified as input. The iteration stops when no more groups can be combined into one. The similarity scores used are the BlockSimilarity scores calculated as follows.

A multiple alignment M of the sequences to be grouped is obtained. Conserved regions are located in M based on the amino acids that appear in all the sequences. If the amino acids are identical or if they are *similar* according to the explanation given earlier for *similarity* considered by DePiCt in any given column of M, then that column is considered as *conserved*. Consecutive conserved columns of M give rise to conserved regions or blocks. The BlockSimilarity score of a block is simply the number of columns in it. If the BlockSimilarity score of a block is $< \lceil k/3 \rceil$ (as the sequences are protein sequences and each amino acid corresponds to three nucleotides in the primer), then it is assigned a score of 0. The BlockSimilarity score of the alignment M is the sum of the BlockSimilarity scores of all the blocks in it.

Two degenerate primers p_f and p_r are designed for each cluster, p_f being the forward primer and p_r the reverse primer, by reverse translating the conserved blocks of the cluster into nucleotide sequences that correspond to the codons of the amino acids in the conserved blocks. For the reverse primers the complement of the nucleotides is considered. The set of all such primers designed is the desired set P_d.

75.1.6.3 Algorithm Multiple Iterative Primer Selector

Proposed by Souvenir et al. [5], algorithm MIPS follows an iterative beam search technique to design degenerate primers. It starts with a set of primers that cover two sequences from an input of n sequences. To bring down the time complexity, the 2-primers are formed only by merging a k-mer with those k-mers that are returned by a technique similar to a FASTA lookup table. Then it extends the coverage of the primers in the candidate set by one additional sequence, introducing degeneracy in the primers if necessary, retains a subset of these primers (the number determined by an input parameter called beam size b) for the next iterative step until none of the primers can be extended further without crossing the target degeneracy. At this point, the primer with the lowest degeneracy is selected and the sequences that it covers (let the number be q) are removed from the input set and the procedure is repeated until all the sequences are covered.

The input dataset for the algorithm is generated as follows. Each input sequence has two sequences representing it in the dataset, one sequence is the first r nucleotides, and the other is the complement of the last r nucleotides of the sequence itself. Thus the input set will consist of $N = 2n$ sequences of length r each. MIPS has an overall time complexity of $O(bN^3rp)$, where b is the beam size, N the number of input sequences, r the sequence length, and p the cardinality of the final set of selected degenerate primers (P_d). The pseudocode of algorithm MIPS is hereunder:

```
Algorithm **MIPS** {
        Let $P$ be the list of selected primers, initially, $P$ is empty;

        Let $Q$ be a priority queue of size $b$ that holds the primer candidates;
        (candidates in $Q$ are ordered with respect to their degeneracy).
        Initially all the $N$ input sequences are alive;
        While (# of sequences alive $> 0$) {
            Let $p_d$ be the selected primer for the current iteration; Initially $p_d$ = null;
            Let $C$ be the collection of all substrings of length $k$ ($k$-mers) in the
            sequences alive;
            For each element $k$-mer $u \in C$ {
               Let $C'$ be the collection of $k$-mers that are obtained from the
               FASTA lookup of $u$;
               For each element $k$-mer $v \in C'$ {
                  Form the primer $u' = u \cup v$;
                  Add $u'$ to $Q$ (if the degeneracy of $u'$ is at most
                  the target degeneracy $d$);
               }
            }
            While( $Q$ is not empty) {
                Let $Q'$ be a priority queue of the next generation candidates;
                For each element $q \in Q$ {
                    For each sequence $S_i$ alive and not covered by $q$ {
                        For each $k$-mer $v$ of $S_i$ {
                            Form the primer $q' = q \cup v$;
                            Add $q'$ to $Q'$ if the degeneracy of $q'$ is at most the
                            target degeneracy $d$;
                        }
                    }
                }
                $p_d$ = the primer of lowest degeneracy in $Q'$;
                $Q = Q'$;
            }
            Add $p_d$ to $P$;
            Set the sequences not covered by $p_d$ as the sequences alive;
        }
        Output $P$;
}
```

Let us analyze the time taken by the loop that processes the elements of the priority queue Q to generate primers of higher coverage. Forming each q' takes $O(N + |\Sigma|k)$ time. Since each candidate can form at most $O(Nr)$ such q', the time complexity of creating Q' is $O(bNr(N + |\Sigma|k))$. Therefore, the time required for one iteration of the loop is $O(bN^2r)$.

Now, let us look into the number of iterations the algorithm will perform to design one primer of degeneracy at most d. The algorithm constructs i-primers in each iteration from $(i-1)$-primer candidates of the previous iteration whose degeneracy either remains the same or increases. Thus, the number of iterations performed by algorithm MIPS to identify one primer of the output set is $O(N)$, leading to a runtime of $O(bN^3r)$. If there are p primers in the output, then, the overall time complexity of algorithm MIPS is $O(bN^3rp)$.

75.1.6.4 Algorithm Degenerate Primer Search (DPS)

An algorithm called DPS has been given in Ref. [13]. DPS has been shown to have a better runtime than that of MIPS in the worst case. It employs a new strategy of ranking the primers in every iteration as defined below.

Definition 75.3

The coverage-efficiency $e(P)$ of a degenerate primer P is the ratio of the number of sequences it amplifies or covers $(c(P))$ to its degeneracy $(d(P))$, that is, $e(P) = c(P)/d(P)$.

Let P_1 and P_2 be two degenerate primers in the priority queue of candidate primers and let $e(P_1) > e(P_2)$. Then the priority of P_1 is higher than that of P_2. If $e(P_1) = e(P_2)$, then the primers are ranked in the nondecreasing order of their degeneracy.

In every iteration, the new algorithm performs additional processing of primer candidates before selecting the b best primers for the next iteration. Instead of adding each q' directly to the priority queue Q', the candidates are collected in a collection B, sorted in their lexicographic order and unique primer candidates are identified by scanning the sorted collection B, obtaining their coverage by merging the coverage of the duplicates. Each such unique primer candidate is added to the priority queue Q', in which the priority of the candidates are as explained above. This ensures that the degeneracy of the candidates generated for $(i + 1)$th iteration from a candidate of ith iteration is strictly greater than that of their predecessor. As the number of symbols that can be added to a nondegenerate primer to create a degenerate primer of degeneracy at most d lies in the range $[\lfloor \log_2 d \rfloor : (|\Sigma| - 1) * \lceil \log_{|\Sigma|} d \rceil]$, the number of iterations the new algorithm performs to identify a single primer of the output set P is $O(|\Sigma| \log_{|\Sigma|} d)$. If $|P| = p$, then the overall time complexity of the algorithm is $O(|\Sigma| \log_{|\Sigma|} dbN^2rp)$, an improvement of the worst-case time complexity $O(bN^3rp)$ of algorithm MIPS.

75.2 The Planted Motif Search Problem

Motif search is an important problem in biology. Motif search is nothing but the problem of identifying short patterns (also called *motifs*) from a database of biological sequences. These motifs are fundamental functional elements in proteins vital for understanding gene function, human disease, and identifying potential therapeutic drug targets. Many variants of motif search have been proposed in the literature. The version of interest in this chapter is defined next.

Inputs are n sequences of length m each. Inputs also are two integers l and d. The problem is to find a motif (i.e., a sequence) M of length l. It is given that each input sequence contains a variant of M. The variants of interest are sequences that are at a hamming distance of d from M. This problem is also known as the *planted* (l, d)*-motif search problem.*

A simple algorithm can be devised for the solution of this problem. Consider every possible l-mer one at a time and check if this l-mer is the correct motif M. There are 4^l possible l-mers. Let M' be one such l-mer. We can check if $M' = M$ as follows. Let the input sequences be $S_1, S_2, \ldots, S_n$. The length of each sequence is m. Form all possible l-mers from out of these sequences. The total number of l-mers is $\leq nm$. Call this collection of l-mers C. Compute the hamming distance between u and M' for every $u \in C$. As a result we can check if M' occurs in each input sequence (at a hamming distance of d). Thus we can identify all the motifs of interest in a total of $O\left(nml4^l\right)$ time. This algorithm becomes impractical even for moderately large values of l. Numerous efficient algorithms have been proposed in the literature.

Algorithms for PMS can be broadly classified into *exact* and *approximate* algorithms. An exact algorithm always outputs the planted motif from a given input of sequences. However, an approximate algorithm may not always output the correct planted motif. Note that this notion of an approximate algorithm differs from the traditional concept of approximation algorithms. The random projection algorithm of Buhler and Tompa [14] is an example of an approximate algorithm and the PMS algorithms given in Ref. [15] are exact.

Algorithms for Problem 1 can be categorized into two depending on the basic approach employed, namely, *profile-based algorithms* and *pattern-based algorithms.* Profile-based algorithms predict the starting positions of the occurrences of the motif in each sequence and pattern-based algorithms predict the motif itself.

Examples of pattern-based algorithms include PROJECTION [14], MULTIPROFILER [16], MITRA [17], and PatternBranching [18]. Examples of profile-based algorithms include CONSENSUS [19], GibbsDNA [20], MEME [21], and ProfileBranching [18]. The performance of profile-based algorithms are specified with a measure called "performance coefficient." The performance coefficient gives an indication of how many positions (for the motif occurrences) have been predicted correctly. These algorithms have been shown to perform well in practice for $l \leq 18$ and $d \leq 6$. A profile-based algorithm could either be approximate or exact. Likewise a pattern-based algorithm may either be exact or approximate.

Several exact algorithms have been proposed in the literature. These algorithms work by exhaustive enumeration. For example, see Refs. [15,22–28]. As pointed out in Ref. [14], these algorithms "become impractical for the sizes involved in the challenge problem." (Challenge problems are instances of the PMS problem that have been found to be difficult and were proposed by Pevzner and Sze [29].) Exceptions are the MITRA algorithm [17] and the PMS algorithms of Rajasekaran et al. [15]. These algorithms are pattern based and are exact. MITRA solves, for example, the (15, 4) instance in 5 min using 100 MB of memory [17]. This algorithm is based on the WINNOWER algorithm [29] and uses pairwise similarity information. A new pruning technique enables MITRA to be more efficient than WINNOWER. MITRA uses a mismatch tree data structure and splits the space of all possible patterns into disjoint subspaces that start with a given prefix. The same (15, 4) instance is solved in 3.5 min by PMS [15].

Profile-based algorithms such as CONSENSUS, GibbsDNA, MEME, and ProfileBranching take much less time for the (15, 4) instance [18]. However, these algorithms fall under the approximate category and may not always output the correct answer. Some of the pattern-based algorithms (such as PROJECTION, MULTIPROFILER, and PatternBranching) also take much less time [18]. However, these are approximate as well (though the success rates are close to 100%).

75.2.1 The WINNOWER Algorithm

The algorithm of Pevzner and Sze [29] (called *WINNOWER*) works as follows. If A and B are two instances (i.e., occurrences) of the motif, then the hamming distance between A and B is at most $2d$. The algorithm constructs a collection C of all possible l-mers in the input. A Graph $G(V, E)$ is then constructed. Each l-mer in C will correspond to a node in G. Two nodes u and v in G are connected by an edge if and only if the hamming distance between the two l-mers is at most $2d$ and these l-mers come from two different sequences.

Clearly, the n instances of the motif M form a clique of size n in G. Thus, the problem of finding M reduces to that of finding large cliques in G. Unfortunately, there will be numerous "spurious" edges (i.e., edges that do not connect instances of M) in G and also finding cliques is $\mathcal{NP}$-hard. Pevzner and Sze [29] employ a clever technique to prune spurious edges. More details can be found in Ref. [29].

75.2.2 Random Projection Algorithm

The algorithm of Buhler and Tompa [14] is based on random projections. Let the motif M of interest be an l-mer. Collect all the l-mers from all the n input sequences and let C be this collection. Project these l-mers along k randomly chosen positions (for some appropriate value of k). In other words, for every l-mer $u \in C$, generate a k-mer u', which is a subsequence of u corresponding to the k random positions chosen. (The random positions are the same for all the l-mers.) We can think of each k-mer thus generated as an integer. We group the k-mers according to their integer values (i.e., we hash all the l-mers using the k-mer of any l-mer as its hash value).

If a hashed group has at least a threshold number s of l-mers in it, then there is a good chance that M will have its k-mer equal to the k-mer of this group. (An appropriate value for s is obtained using a probabilistic analysis.) We collect all the k-mers (and the corresponding l-mers) that pass the threshold and these are processed further to arrive at the final answer M. Processing is done using the expectation maximization (EM) technique of Lawrence and Reilly [30].

75.2.3 Algorithm PMS1

A simple algorithm called PMS1 has been given in Ref. [15]. Even this simple algorithm has been shown to solve some of the challenge problems efficiently. Steps involved in this algorithm are:

1. Generate all possible l-mers from out of each of the n input sequences. Let C_i be the collection of l-mers from out of S_i for $1 \leq i \leq n$.
2. For all $1 \leq i \leq n$ and for all $u \in C_i$ generate all l-mers v such that u and v are at a hamming distance of d. Let the collection of l-mers corresponding to C_i be C'_i, for $1 \leq i \leq n$. The total number of patterns in any C'_i is $O\left(m\binom{l}{d}3^d\right)$.
3. Sort all the l-mers in every C'_i, $1 \leq i \leq n$ and eliminate duplicates in every C'_i. Let L_i be the resultant sorted list corresponding to C'_i.
4. Merge all the L_is $(1 \leq i \leq n)$ and output the generated (in step 2) l-mer that occurs in all the L_is.

The run time of the above algorithm is $O\left(nm\binom{l}{d}3^d\frac{l}{w}\right)$ where w is the word length of the computer.

Two other exact algorithms called PMS2 and PMS3 have also been proposed in Ref. [15]. These algorithms are competitive in practice with other exact algorithms. For a survey on motif search algorithms see Ref. [31].

75.3 Closest String and Closest Substring Problems

Two problems that are closely related to the PMS are the closest string problem (CSP) and the closest substring problem (CSSP). The CSP takes as input n sequences of length m each and the problem is to identify a string s of length m that is the closest to all the input strings. In other words, the maximum distance of s to any input sequence should be minimum. However, the CSSP takes as input n sequences of length m each. The problem is to identify a string $\bar{s}$ of length $l(<m)$ such that $\bar{s}$ is the closest to some substrings (each of length l and picked one from each input sequence) of the input sequences.

Algorithms that have been proposed for the PMS problem have typically been tested on random inputs. Specifically, the input sequences will be generated randomly such that each symbol in each sequence is uniformly randomly picked from Σ. A motif M will also be generated randomly in a similar fashion. This motif will then be planted in the input sequences starting from random locations. What is planted in each sequence will be a random neighbor of M that is at a Hamming distance of $\leq d$ from M. If l is small enough in relation to d, then there could be spurious motifs occurring in the input sequences by random chance (see, e.g., Ref. [14]). A probabilistic analysis can be performed to figure out values of l and d for which the probability of a spurious motif occurring by random chance is very low. For these values of l and d, in fact the planted motif will correspond to the closest substring. If there are spurious motifs in the input sequences then the closest substring may not be the same as the planted motif. However, in this case it may not be possible to identify the planted motif using any other algorithm also unless additional information is given for the planted motif. For example, the exact algorithms of [15] will identify all the motifs present in the input (including the planted motif). But the algorithm will not be able to isolate the planted motif.

Both CSP and CSSP have been proven to be NP-hard. In this section we present some of the approximation algorithms that have been devised for CSP and CSSP.

75.3.1 The Closest String Problem

In this section we address the CSP. A formal definition of the CSP follows.

Definition 75.4

Given strings $s_1, s_2, \ldots, s_n$ (of length m each) over the alphabet Σ, the CSP *is to find a string s of length m over Σ that minimizes $\max_{i=1}^{n} d(s, s_i)$, where d is the Hamming distance. Let $d_{min} := \max_{i=1}^{n} d(s, s_i)$.*

From hereon whenever we refer to $s_1, \ldots, s_n$ we assume that they are strings of length m over the alphabet Σ.

The first result on the complexity of the problem was obtained in Ref. [32] under the context of coding theory and the so-called *minimum radius problem* and states the following.

Theorem 75.1

If $\Sigma = \{0, 1\}$ the CSP *is NP-complete.*

After this negative result, two different approaches were used to tackle this problem. The first approach had the goal of finding polynomial-approximation schemes with a prescribed accuracy (see, e.g., [33–35]). The other approach sought to find exact solutions that take polynomial time for a fixed set of parameters (such as n or d_{min}). Examples include Refs. [36,37].

In the next sections we will describe the first approach. The interested reader can find details of the second in Refs. [36,37].

75.3.1.1 Simple Approximation Algorithms

Given any instance of the *CSP*, one of the easiest strategies is to output any of the given strings. This gives rise to the following theorem due to Ref. [33].

Theorem 75.2

Fix $1 \leq i \leq n$. The algorithm that outputs s_i is a 2-approximation algorithm for the CSP.

Proof

Let us call s the optimal solution to the CSP and let $d_{min} := \max_{i=1}^{n} d(s, s_i)$. Given $1 \leq j \leq n$ we have that

$$d(s_i, s_j) \leq d(s_i, s) + d(s, s_j) \leq 2d_{min}$$

Hence we have $\max_{j=1}^{n} d(s_i, s_j) \leq 2d_{min}$ and the result follows. □

The following result from Ref. [35] will be used later on, and will allow us to find an exact solution when $m \leq c \log n$ for a fixed c.

Theorem 75.3

There is a polynomial-time algorithm that solves the CSP when $m \leq c \log n$.

Proof

We proceed by enumerating all of the strings of length m and picking the one that minimizes the desired distance. Since $|\Sigma|^m \leq |\Sigma|^{c \log n} = n^{c'}$ (for some constant c') we have that it takes polynomial time in n.

□

75.3.1.2 An Approximate Solution Using Integer Programming

A useful and widely used strategy in approximation algorithms is the so-called "method of randomized rounding" [38]. This method models the given problem as a linear integer program, relaxes the integrality constraints, solves the resultant problem by a polynomial-time linear programming solver and, rounds the—possible—real solution to an integer solution based on the values obtained. This strategy was first used in Ref. [33].

We could use the above strategy to solve the CSP as well.

Definition 75.5

Given a string $p = p[1] \ldots p[m]$ over an alphabet Σ we define the following binary variables, for $\sigma \in \Sigma$ and $i = 1, \ldots, m$: $p_i^{\sigma} = \begin{cases} 1 & \textit{if } p[i] = \sigma \\ 0 & \textit{if not} \end{cases}$

Note that given a set of binary variables $\{s_i^{\sigma}\}$, where $i = 1, \ldots, m$ and $\sigma \in \Sigma$ they represent a string of length m if for every $i = 1, \ldots, m$ there is exactly one 1 in the sequence $\{s_i^{\sigma}\}_{\sigma \in \Sigma}$ or equivalently if $\sum_{\sigma \in \Sigma} s_i^{\sigma} = 1$.

We would like to find a formula that will allow us to calculate the hamming distance between two strings x and y by using the binary variables x_i^{σ} and y_i^{σ}. To do that we introduce the following definition and lemma.

Definition 75.6

Given two strings $p = p[1] \cdots p[m]$ and $q = q[1] \cdots q[m]$, for any i ($1 \leq i \leq m$) we define $\delta_i(p, q) = \begin{cases} 1 & \textit{if } p[i] \neq q[i] \\ 0 & \textit{if } p[i] = q[i] \end{cases}$

Lemma 75.1

Given strings $p = p[1] \cdots p[m]$ and $q = q[1] \cdots q[m]$ we have

$$d(p, q) = \sum_{i=1}^{m} \delta_i(p, q) = \sum_{i=1}^{m} \left(1 - \sum_{\sigma \in \Sigma} p_i^\sigma q_i^\sigma \right)$$

Proof

This follows easily from the fact that $p_i^\sigma q_i^\sigma = \begin{cases} 1 & \text{if } p[i] = q[i] = \sigma \\ 0 & \text{otherwise} \end{cases}$. □

Notice that if one of the sequences is fixed, the equation obtained in lemma 75.1 is linear.

To state the CSP as a minimization problem suppose that $s_i = s_i[1], \ldots, s_i[m]$ for $i = 1, \ldots, n$. The problem can be stated as

$$\begin{aligned} &\min \max_{i=1}^{n} \; d(r, s_i) \\ &\sum_{\sigma \in \Sigma} r_i^\sigma = 1, \quad i = 1, \ldots, n \\ &r_i^\sigma \in \{0, 1\}, \quad i = 1, \ldots, n \text{ and } \sigma \in \Sigma \end{aligned} \tag{75.1}$$

Let d represent the maximum distance of r to any s_i. Using lemma 75.1 we get the following linear integer program:

$$\begin{aligned} &\min \; d \\ &\sum_{j=1}^{m} \left(1 - \sum_{\sigma \in \Sigma} s_{i,j}^\sigma r_j^\sigma \right) \le d, && i = 1, \ldots, n \\ &\sum_{\sigma \in \Sigma} r_i^\sigma = 1, && i = 1, \ldots, n \\ &r_i^\sigma \in \{0, 1\}, && i = 1, \ldots, n \text{ and } \sigma \in \Sigma \end{aligned} \tag{75.2}$$

By allowing the variables to take values in the interval $[0, 1]$, we get the following linear program:

$$\begin{aligned} &\min \; d \\ &\sum_{j=1}^{m} \left(1 - \sum_{\sigma \in \Sigma} s_{i,j}^\sigma r_j^\sigma \right) \le d, && i = 1, \ldots, n \\ &\sum_{\sigma \in \Sigma} r_i^\sigma = 1, && i = 1, \ldots, n \\ &0 \le r_i^\sigma \le 1, && i = 1, \ldots, n \text{ and } \sigma \in \Sigma \end{aligned} \tag{75.3}$$

We will call $\hat{d}_{min}$ the solution to Eq. (75.3) and we will call $\bar{r}_i^\sigma$ the values that the variables r_i^σ take for $i = 1, \ldots, m$ and $\sigma \in \Sigma$. It is clear that $\hat{d}_{min} \le d_{min}$.

Definition 75.7

1. *Given $\bar{r}_i^\sigma$ with $i = 1, \ldots, m$ and $\sigma \in \Sigma$, which are solutions to Eq.* (75.3), *we define random variables x_i independently for $i = 1, \ldots, m$ by satisfying the equation* $\Pr(\{x_i = \sigma\}) = \bar{r}_i^\sigma$ *for $\sigma \in \Sigma$. Note that* $\sum_{\sigma \in \Sigma} \Pr(\{x_i = \sigma\}) = \sum_{\sigma \in \Sigma} \bar{r}_i^\sigma = 1$
2. *Let x be the string obtained by concatenating the $\{x_i\}_{i=1}^m$, that is, $x = x_1 x_2 \cdots x_m$.*
3. *We call $d_x = \max_{i=1}^m \; d(x, s_i)$*

It is clear that one can obtain x by a polynomial-time algorithm by solving the linear programming problem (75.3) and then do the randomized rounding described in Definition (75.7). We will prove now that x is a good approximation to the solution.

Lemma 75.2

$E[d(x, s_i)] \le d_{min}$ for $i = 1, \ldots, n$.

Proof

By Lemma 75.1 we have that $d(x, s_i) = \sum_{j=1}^{m} \delta_j(x, s_i)$. Furthermore for fixed i (in the range $[1, m]$) and j (in the range $[1, n]$), $\delta_i(x_i, s_j)$ is a Bernoulli trial with

$$\mathrm{E}[\delta_j(x, s_i)] = 1 - \sum_{\sigma\in\Sigma} s^{\sigma}_{i,j}\mathrm{E}[x^{\sigma}_j] = 1 - \sum_{\sigma\in\Sigma} s^{\sigma}_{i,j}\,\Pr(\{x_j = \sigma\}) = 1 - \sum_{\sigma\in\Sigma} s^{\sigma}_{i,j}\tilde{r}^{\sigma}_j$$

Hence by linearity of expectations we get that $\mathrm{E}[d(x, s_i)] = \sum_{j=1}^{m}(1 - \sum_{\sigma\in\Sigma} s^{\sigma}_{i,j}\tilde{r}^{\sigma}_j))$ and since r^{σ}_j is a solution of Eq. (75.3) we have

$$\mathrm{E}[d(x, s_i)] = \sum_{j=1}^{m}\left(1 - \sum_{\sigma\in\Sigma} s^{\sigma}_{i,j}\tilde{r}^{\sigma}_j\right) \le \hat{d}_{min} \le d_{min} \quad \text{for } i = 1, \ldots, n \qquad \square$$

We can employ Chernoff bounds to prove a stronger result.

Lemma 75.3

Let $Y_1, Y_2, \ldots, Y_n$ be independent Bernoulli trials with $E(Y_i) = p_i$. If $Y = \sum_{i=1}^{n} Y_i$ and $\mu = E(Y) = \sum_{i=1}^{n} p_i$ and $0 < \epsilon \le 1$ we have

$$\Pr(Y \ge (1+\epsilon)\mu) \le e^{-\frac{1}{3}\mu\epsilon^2} \quad \Pr(Y \ge \mu + \epsilon n) \le e^{-\frac{1}{3}n\epsilon^2}$$
$$\Pr(Y \le (1+\epsilon)\mu) \le e^{-\frac{1}{2}\mu\epsilon^2} \quad \Pr(Y \le \mu - \epsilon n) \le e^{-\frac{1}{2}n\epsilon^2}$$

Theorem 75.4

The algorithm that outputs x is a randomized polynomial-time $(1+\epsilon)$ approximation algorithm for the CSP *when $d_{min} \ge \frac{6\log n}{\epsilon^2}$.*

Proof

We have that $d(x, s_i)$ is the sum of independent Bernoulli trials, that is, $d(x, s_i) = \sum_{j=1}^{m} \delta_j(x, s_i)$, and by Lemma 75.2 we have that $\mu = E(d(X, s_i)) \le d_{min}$. By applying Lemma 75.3 we have

$$\Pr\left(d(x, s_i) > (1+\epsilon)d_{min}\right) \le e^{-\frac{1}{3}\epsilon^2\mu}$$

Furthermore we have

$$\Pr(d_x > (1+\epsilon)d_{min}) = \Pr(\{\forall i = 1, \ldots, n : d(X, s_i) > (1+\epsilon)d_{min}\}) \le ne^{-\frac{1}{3}\epsilon^2\mu}$$

And since $\mu \ge d_{min} \ge \frac{6\log n}{\epsilon^2}$ we have

$$\Pr\left(d_x > (1+\epsilon)d_{min}\right) \le ne^{-2logn} \le \frac{1}{n} \qquad \square$$

Using the method of conditional probabilities (see Ref. [38]) it is possible to derandomize the previous algorithm. This is done explicitly in Refs. [34,35].

75.3.1.3 A $(1+\epsilon)$ Polynomial-Approximate Scheme

In this section we describe the approximation scheme of Refs. [34,35]. Consider the following strategy to solve the CSP, fix $0 \le k < n$ and align any k strings out of the n. In this alignment, there will be *clean* columns and *dirty* columns. A column is clean if a single character occurs in the entire column; it is dirty otherwise. If j is a clean column and c is the character in this column, then in the output string column j will be set to c. Luckily the number of dirty columns will be relatively small and can be dealt with using the methods introduced in the previous section.

In the remainder of this section we assume a fixed k, such that $1 < k < n$ and we make the previous ideas rigorous in the following way.

Definition 75.8

Let $i_1, \ldots, i_k$ be a subset of indices of $\{1, \ldots n\}$. We define:

1. *$Q_{i_1,\ldots,i_k} := \{j : s_{i_1}[j] = s_{i_2}[j] = \ldots s_{i_k}[j]\}$, that is, the set of positions where $s_{i_1}, \ldots, s_{i_k}$ agree.*
2. *$P_{i_1,\ldots,i_k} := \{1, \ldots, m\} \backslash Q_{i_1,\ldots,i_k}$, that is, the set of positions where $s_{i_1}, \ldots, s_{i_k}$ have two or more characters.*
3. *Given a string r of length m over Σ, we define $r|_{\{i_1 \ldots i_k\}} = r[i_1] \ldots r[i_k]$.*

The following lemma gives us a good estimate for the number of dirty columns.

Lemma 75.4

Given $1 \leq i_1 \leq \cdots \leq i_k \leq n$ we have that

$$|P_{i_1 \cdots i_k}| \leq kd_{min}$$

Proof

Let j be a position (i.e., a column) where $s_{i_1}, \ldots, s_{i_k}$ have two or more characters. Then, there is an l $(1 \leq l \leq k)$ such that $s[j] \neq s_{i_l}[j]$, where s is the closest to all the n input strings. By definition, $d(s, s_{i_l}) \leq d_{min}$, and hence every s_{i_l} contributes at most d_{min} dirty columns to $P_{i_1,\ldots,i_k}$ hence $|P_{i_1 \cdots i_k}| \leq kd_{min}$. □

Theorem 75.5 gives us an effective way to find an approximate solution to the CSP of $s_1, \ldots, s_n$ when we restrict every string to the positions $P_{i_1,\ldots,i_k}$ (i.e., the dirty columns).

Theorem 75.5

Let $i_1, \ldots, i_k$ form a subset of indices from $\{1, \ldots, n\}$ and let $0 < \epsilon < 1$. There is a polynomial-time algorithm, which produces a string s' of length $|P_{i_1,\ldots,i_k}|$ such that

$$d(s', s_l|_{P_{i_1,\ldots,i_k}}) \leq (1+\epsilon)d_{min} - d(s_{i_1}|_{Q_{i_1 \ldots i_k}}, s_l|_{Q_{i_1 \ldots i_k}}) \textit{ for } l = 1, \ldots, n$$

Proof

Consider $\tilde{s}_1, \ldots, \tilde{s}_n$, where $\tilde{s}_l[j] : = \begin{cases} s_l[j] & \text{when } j \in P_{i_1,\ldots,i_k} \\ s_{i_1}[j] & \text{when } j \in Q_{i_1,\ldots,i_k} \end{cases}$

Let $\tilde{s}$ be a solution to the CSP over these strings, and define as before $\tilde{d}_{min}$ and $\tilde{P}_{i_1,\ldots,i_k}$. It is simple to note that $\tilde{d}_{min} \leq d_{min}$ and that $\tilde{P}_{i_1,\ldots,i_k} = P_{i_1,\ldots,i_k}$. Hence by using Lemma 75.4 we infer that $|P_{i_1 \ldots i_k}| = |\tilde{P}_{i_1 \ldots i_k}| \leq k\tilde{d}_{min}$.

If $|P_{i_1 \ldots i_k}| > \frac{6k \log n}{\epsilon^2}$ we have that $\tilde{d}_{min} > \frac{6 \log n}{\epsilon^2}$ and by using Theorem 75.4 on $\tilde{s}_1, \ldots, \tilde{s}_j$ we obtain $\tilde{s}'$, such that

$$d(\tilde{s}', \tilde{s}_l) \leq (1+\epsilon)\tilde{d}_{min} \leq (1+\epsilon)d_{min} \quad \text{for } l = 1, \ldots, n$$

Let $s' = \tilde{s}'|_{P_{i_1,\ldots,i_k}}$ and since $\tilde{s}'|_{Q_{i_1,\ldots,i_k}} = s_{i_1}|_{Q_{i_1,\ldots,i_k}}$. Then, that

$$d(\tilde{s}', \tilde{s}_l) = d(s', s_l|_{P_{i_1,\ldots,i_k}}) + d(s_{i_1}|_{Q_{i_1,\ldots,i_k}}, s_l|_{Q_{i_1,\ldots,i_k}}) \leq (1+\epsilon)d_{min}$$

If $|P_{i_1 \ldots i_k}| < \frac{6k \log n}{\epsilon^2}$ define $s'_l := s_l|_{P_{i_1,\ldots,i_k}}$ for $l = 1, \ldots, n$. We can find s' that solves exactly the CSP on $s'_1, \ldots s'_n$ using Theorem 75.3, considering that $|s'_l| = |P_{i_1 \ldots i_k}| < c \log n$.

If we define $\tilde{s}$ by $\tilde{s}[j] : = \begin{cases} s'[j] & \text{when} j \in P_{i_1,\ldots,i_k} \\ s_{i_1}[j] & \text{when} j \in Q_{i_1,\ldots,i_k} \end{cases}$ it is clear that this is a solution to the CSP for $\tilde{s}_1, \ldots, \tilde{s}_n$. Also,

$$d(\tilde{s}, \tilde{s}_l) = d\big(s', s_l|_{P_{i_1,\ldots,i_k}}\big) + d\big(s_{i_1}|_{Q_{i_1,\ldots,i_k}}, s_l|_{Q_{i_1,\ldots,i_k}}\big) \leq \tilde{d}_{min} \leq d_{min}$$ □

For the remaining part of this section we will be interested in knowing how close is s_{i_1} to s when we restrict the problem to the positions $Q_{i_1,\ldots,i_k}$ (i.e., the clean columns). That is, we want to estimate

$$d\left(s_l\big|_{Q_{i_1\ldots i_k}}, s_{i_1}\big|_{Q_{i_1\ldots i_k}}\right) - d\left(s_l\big|_{Q_{i_1\ldots i_k}}, s\big|_{Q_{i_1\ldots i_k}}\right) \quad \text{for } l = 1, \ldots, n$$

The following definition and lemma will be a step in that direction.

Definition 75.9

Let $i_1, \ldots, i_k$ be a subset of indices of $\{1, \ldots n\}$ and $1 \le l \le n$, we define

$$J(l) = \{j \in Q_{i_1\ldots i_k} : s_{i_1}[j] \neq s_l[j] \wedge s_{i_1}[j] \neq s[j]\}$$

Lemma 75.5

Let $i_1, \ldots, i_k$ be a subset of indices of $\{1, \ldots n\}$ and $1 \le i \le n$, then

$$d\left(s_l\big|_{Q_{i_1\ldots i_k}}, s_{i_1}\big|_{Q_{i_1\ldots i_k}}\right) - d\left(s_l\big|_{Q_{i_1\ldots i_k}}, s\big|_{Q_{i_1\ldots i_k}}\right) \le J(l) \tag{75.4}$$

Proof

Let $Q := Q_{i_1,\ldots,i_k}$. Then,

$$\begin{aligned}
& d(s_l|_Q, s_{i_1}|_Q) - d(s_l|_Q, s|_Q) \\
= \ & |\{j \in Q : s_{i_1}[j] \neq s_l[j]\}| - \{j \in Q : s_l[j] \neq s[j]\}| \\
\le \ & |\{j \in Q : s_{i_1}[j] \neq s_l[j] \wedge s_l[j] = s[j]\}| + (|\{j \in Q : s_{i_1}[j] \neq s_l[j] \wedge s_l[j] \neq s[j]\}| \\
& - |\{j \in Q : s_l[j] \neq s[j]\}|) \\
\le \ & |\{j \in Q : s_{i_1}[j] \neq s_l[j] \wedge s_{i_1}[j] \neq s[j]\}| + (|\{j \in Q : s_l[j] \neq s[j]\}| \\
& - |\{j \in Q : s_l[j] \neq s[j]\}|) \le J(l)
\end{aligned}$$

□

Note that for a fixed k, $J(l)$ depends on the set of indices $i_1, \ldots, i_k$ that we choose. For an arbitrary set of indices $i_1, \ldots, i_k$ we cannot bound $J(l)$, but we will prove in the following lemmas that there exists a set of indices $i_1, \ldots, i_k$, where $J(l)$ is small. By using Lemma 75.5 we know that s_{i_1} is an *approximate solution* for the restriction of the problem to the positions $Q_{i_1,\ldots,i_k}$. To be more precise we introduce the following.

Definition 75.10

Let $i_1, \ldots, i_k$ be a subset of indices from $\{1, \ldots, n\}$ and $0 \le l \le n$, we define

1. $p_{i_1,\ldots,i_k} := d(s_{i_1}\big|_{Q_{i_1,\ldots,i_k}}, s\big|_{Q_{i_1,\ldots,i_k}})$ *that is, the number of mismatches between s_{i_1} and s at positions in $Q_{i_1,\ldots,i_k}$.*
2. $\rho_0 := \max\limits_{1 \le i, j \le n} \frac{d(s_i, s_j)}{d_{min}}$ *and* $\rho_k := \min\limits_{1 \le i_1 \le \cdots i_k \le n} \frac{p_{i_1 \cdots i_k}}{d_{min}}$ *for $k = 1, \ldots, n$.*

Lemma 75.6

For any $2 \le k' \le k$ there are indices $1 \le i_1 \le \cdots \le i_k \le n$ such that for any $1 \le l \le n$

$$J(l) \le (\rho_{k'} - \rho_{k'+1}) d_{min}$$

Proof

Choose $i_1, \ldots, i_{k'}$ such that $p_{i_1 \cdots i_{k'}} = \rho_{k'} d_{min}$. Then for any $1 \le i_{k'+1} \le i_{k'+2} \le \cdots \le i_k \le n$ and $1 \le l \le n$ we have that

$$\begin{aligned}
J(l) & \le |\{j \in Q_{i_1 \cdots i_{k'}} : s_{i_1}[j] \neq s_l[j] \wedge s_{i_1}[j] \neq s[j]\}| \\
& = |\{j \in Q_{i_1 \cdots i_{k'}} : s_{i_1}[j] \neq s[j]\} \setminus \{j \in Q_{i_1 \cdots i_{k'}} : s_{i_1}[j] = s_l[j] \wedge s_{i_1}[j] \neq s[j]\}| \\
& = |\{j \in Q_{i_1 \cdots i_{k'}} : s_{i_1}[j] \neq s[j]\}| - |\{j \in Q_{i_1 \cdots i_{k'}, l} : s_{i_1}[j] \neq s[j]\}| \\
& = p_{i_1 \ldots i_{k'}} - p_{i_1 \ldots i_{k'}, l} \le (\rho_{k'} - \rho_{k'+1}) d_{min}
\end{aligned}$$

□

Lemma 75.7

For $2 \le k < n$

$$\min\{\rho_0 - 1, \rho_2 - \rho_3, \ldots, \rho_k - \rho_{k+1}\} \le \frac{1}{2k-1}$$

Proof

$$\frac{1}{2}(\rho_0 - 1) + (\rho_2 - \rho_3) + \cdots + (\rho_k - \rho_{k+1}) = \frac{1}{2}(\rho_0 - 1) + \rho_2 - \frac{1}{2} - \rho_{k+1} \le \frac{1}{2}\rho_0 + \rho_2 - \frac{1}{2}$$

However, let i, j be such that $d(s_i, s_j) = \rho_0 d_{min}$. Then, in the positions where s_i differs from s_j, one of the two strings say s_i is at a distance of at least $\frac{\rho_0}{2} d_{min}$ from the optimum, that is, $d(s|_{P_{i,j}}, s_i|_{P_{i,j}}) \ge \frac{\rho_0}{2} d_{min}$. Thus,

$$d(s|_{Q_{i,j}}, s_i|_{Q_{i,j}}) \le d_{min} - \frac{\rho_0}{2} d_{min} = (1 - \rho_0) d_{min}$$

This implies that $\rho_2 \le (1 - \frac{\rho_0}{2})$ and as a consequence,

$$\frac{\frac{1}{2}(\rho_0 - 1) + (\rho_2 - \rho_3) + \cdots + (\rho_k - \rho_{k+1})}{k - 1 + \frac{1}{2}} \le \frac{\frac{1}{2}}{k - \frac{1}{2}} = \frac{1}{2k-1}$$

So at least one of $\rho_0 - 1, \rho_2 - \rho_3, \ldots, \rho_k - \rho_{k+1}$ is less than or equal to $\frac{1}{2k-1}$. □

Theorem 75.6

There exists a set of indices $1 \le i_1 \le \cdots \le i_k \le n$ *such that*

$$d\left(s_l\big|_{Q_{i_1\cdots i_k}}, s_{i_1}\big|_{Q_{i_1\cdots i_k}}\right) - d\left(s_l\big|_{Q_{i_1\cdots i_k}}, s\big|_{Q_{i_1\cdots i_k}}\right) \le \frac{1}{2k-1} d_{min} \textit{ for } 1 \le l \le n$$

Proof

It is clear by using Lemmas 75.5, 75.6, and 75.7 in consecutive order. □

Based on the ideas presented consider the following algorithm.

Algorithm Closest_String

```
1. for every set of indices $\{i_1, \ldots, i_k\}$ do
   (a) Let $\hat{s}(i_1, \ldots, i_k)$ be the solution to the problem as in Theorem 75.5.
   (b) Define $s(i_1, \ldots, i_k)$ by making $s(i_1, \ldots, i_k)|_{Q_{i_1,\ldots,i_k}} := s_{i_1}|_{Q_{i_1,\ldots,i_k}}$ and
       $s(i_1, \ldots, i_k)|_{P_{i_1,\ldots,i_k}} := \hat{s}(i_1, \ldots, i_k)$.
   (c) Let $cost(i_1, \ldots, i_k) := \max_{j=1}^{k} d(s_{i_j}, s(i_1, \ldots, i_k))$.
2. Let $s' := s(i_1, \ldots, i_k)$ be the string that minimizes $cost(i_1, \ldots, i_k)$.
3. for i= $1, \ldots, n$ do calculate $cost(i) := \max_{j=1}^{n} d(s_j, s_i)$.
4. Select the string of minimum cost from the two previous steps.
```

Theorem 75.7

Let $0 < \delta < 1$. *The algorithm* **Closest_String** *is a* $(1 + \delta)$ *polynomial-approximation algorithm for the CSP.*

Proof

Choose $1 < k < n$ and $0 < \epsilon < 1$ such that $\frac{1}{2k-1} + \epsilon \le \delta$.

If $\rho_0 - 1 \le \frac{1}{2k-1}$ then in step 2 we find a solution such that

$$\rho_0 d_{min} \le \left(1 + \frac{1}{2k-1}\right) d_{min}$$

which by definition of ρ_0 implies that

$$d(s', s_l) \le \max_{1 \le i, j \le n} d(s_i, s_j) \le \frac{1}{2k-1} d_{min} \le (1+\delta) d_{min}$$

In case $\rho_0 - 1 > \frac{1}{2k-1} d_{min}$, let us first observe that in step 2 we find a set of indices $1 \le i_1 \le \cdots \le i_k \le n$ such that

$$d(s', s_l) = d\left(s_{i_1}\big|_{Q_{i_1,\ldots,i_k}}, s_l\big|_{Q_{i_1,\ldots,i_k}}\right) + d\left(\hat{s}\left(i_1, \ldots, i_k\right), s_l\big|_{P_{i_1,\ldots,i_k}}\right)$$

Using Theorems 75.5 and 75.6 we note that

$$\begin{aligned} d(s', s_l) &\le \left(1 + \frac{1}{2k-1}\right) d_{min} + d\left(s_{i_1}\big|_{Q_{i_1,\ldots,i_k}}, s_l\big|_{Q_{i_1,\ldots,i_k}}\right) + \epsilon d_{min} - d\left(s_{i_1}\big|_{Q_{i_1,\ldots,i_k}}, s_l\big|_{Q_{i_1,\ldots,i_k}}\right) \\ &\le (1+\delta) d_{min} \end{aligned}$$ □

75.3.2 Closest Substring Problem

The CSSP takes as input n sequences of length m each. The goal is to find a substring (of length l) that is the closest to some substrings (of length l each) picked one from each input sequence. The substring of interest is also known as the "motif." The notion of a motif is defined rigorously next.

Definition 75.11

If s and s' are strings over the alphabet Σ and l is such that $0 < l < |s|$, we define:

1. *$s' \lhd_l s$ if s' is a substring of length l of s.*
2. *$\bar{d}(s', s) := \min_{r \lhd_l s} d(s', r)$.*

Note that if $s' \lhd_l s$ then $\bar{d}(s', s) = 0$.

Definition 75.12

Given strings $s_1, s_2, \ldots, s_n$ of length m each over the alphabet Σ and an l $(0 < l \le m)$, the CSSP *is to find a string $\bar{s}$ of length l over Σ that minimizes $\max_{i=1}^{n} \bar{d}(s, s_i)$. We denote by $t_i \lhd_l s_i$ the string such that $\bar{d}(\bar{s}, s_i) = d(S, \bar{t}_i)$ for $i = 1, \ldots, n$. We denote by $\bar{d}_{min} := \max_{i=1}^{n} \bar{d}(\bar{s}, s_i)$.*

In the remaining part of this section we will assume that we are given $s_1, \ldots, s_n$ and l that satisfy the conditions of Definition 75.12. $\bar{d}_{min}$, $\bar{t}_i$ and $\bar{s}$ will satisfy the conditions stated in Definition 75.12.

We now present approximation strategies that have been proposed in Refs. [35,39,40]. Some of these strategies will be based on the results which were discussed in Section 75.3.

75.3.2.1 Simple Approximation Schemes

We start by presenting a simple strategy that obtains a 2-approximation polynomial-time algorithm as it is described in Ref. [39].

Algorithm Simple_Closest_Substring

```
1. for every $s' \lhd_l s_1$ do
   (a) for $i := 2, \ldots, n$ do
           Let $t_i(s') \lhd_l s_i$ be such that $d(s', t_i(s')) = \bar{d}(s', s_i)$.
   (b) Let $cost(s') := \max_{i=1}^{n} \bar{d}(s', s_i)$.
2. Pick the string $\tilde{s} \lhd_l s_1$ that minimizes $cost(\tilde{s})$. Let $\tilde{t}_j \lhd_l s_j$ be such that
   $d(\tilde{s}, \tilde{t}_j) = \bar{d}(\tilde{s}, s_j)$ for $j = 1, \ldots, n$.
```

Theorem 75.8

Algorithm **Simple_Closest_Substring** *is 2-approximate for CSSP.*

Proof

Let $r_1, \ldots, r_n$, where $r_i \lhd_l s_i$, be the strings such that $\bar{d}(\bar{s}, s_i) = d(\bar{s}, r_i)$. Then, $d(\bar{s}, r_i) \leq \bar{d}_{min}$ and hence

$$d(r_1, r_j) \leq d(\bar{s}, r_1) + d(\bar{s}, r_j) \leq 2\bar{d}_{min} \text{ for } j = 1, \ldots, n$$

We conclude by noticing that

$$\bar{d}(\tilde{s}, s_i) = d(\tilde{t}_1, \tilde{t}_i) = \min_{s' \lhd_l s_1} \max_{j=1}^{n} \; d(s', t_j(s')) \leq \min_{s' \lhd_l s_1} \max_{j=1}^{n} \; d(s', r_j) = \max_{j=1}^{n} \; d(r_1, r_j) \leq 2\bar{d}_{min} \qquad \square$$

Our aim is to describe an approximation algorithm that follows the ideas of algorithm **Closest_String**. To do so we will describe some definitions and theorems that generalize the ones done in Section 75.3.1.3.

Definition 75.13

Given a set of indices $1 \leq i_1 \leq \cdots \leq i_k \leq n$ *and a set of substrings* $T = \{t_{i_1}, \ldots, t_{i_k}\}$ *where* $t_{i_j} \lhd_l s_{i_j}$ *for* $j = 1, \ldots, k$. *We define*

1. $Q^T_{i_1,\ldots,i_k} := \{j : t_{i_1}[j] = t_{i_2}[j] = \cdots t_{i_k}[j]\}$, *that is, the set of positions where* $t_{i_1}, \ldots, t_{i_k}$ *agree.*
2. $P^T_{i_1,\ldots,i_k} := \{1, \ldots, m\} - Q^T_{i_1,\ldots,i_k}$, *that is, the set of positions where* $t_{i_1}, \ldots, t_{i_k}$ *differ.*

The following theorems extend naturally Lemma 75.4 and Theorem 75.6.

Theorem 75.9

For $1 \leq i_1 \leq \cdots \leq i_k \leq n$ *and* $T := \{t_{i_1}, \ldots, t_{i_k}\}$ *where* $t_{i_j} \lhd s_j$,

$$\left|P^T_{i_1 \cdots i_k}\right| \leq k\bar{d}_{min}.$$

Proof

It follows directly from Lemma 75.4. □

Theorem 75.10

There exists a set of indices $1 \leq i_1 \leq \cdots \leq i_k \leq n$ *such that for* $T := \{\bar{t}_{i_1}, \ldots, \bar{t}_{i_k}\}$,

$$d\left(\bar{t}_l\big|_{Q^T_{i_1 \cdots i_k}}, \bar{t}_{i_1}\big|_{Q^T_{i_1 \cdots i_k}}\right) - d\left(\bar{t}_l\big|_{Q^T_{i_1 \cdots i_k}}, \bar{s}\big|_{Q^T_{i_1 \cdots i_k}}\right) \leq \frac{1}{2k-1}\bar{d}_{min} \textit{ for } 1 \leq l \leq n.$$

Proof

It follows from Theorem 75.6 □

Algorithm Closest_Small_Substring

```
1. for every set of substrings $T = \{t_{i_1}, \ldots, t_{i_k}\}$ where $t_{i_j} \lhd_l s_{i_j}$  $j = 1, \ldots, k$ do
   (a) for every $t \in \Sigma^p$ where $p = |P^T_{i_1,\ldots,i_k}|$ do
           Build the string $x$ by $x|_{P^T_{i_1,\ldots,i_k}} = t$ and $x|_{Q^T_{i_1,\ldots,i_k}} = s_{i_1}|_{Q^T_{i_1,\ldots,i_k}}$.
   (b) Let $x^T_{i_1,\ldots i_k}$ be the string that minimizes $\max_{i=1}^n \; d(x^T_{i_1,\ldots i_k}, s_i)$ and call
       $cost^T(i_1, \ldots, i_k) := \max_{i=1}^n \; \bar{d}(x^T_{i_1,\ldots i_k}, s_i)$.
2. Let $x'$ be the string that minimizes $cost^T(i_1, \ldots, i_k)$.
3. for i=:1, ..., $n$ and for every $r \lhd_l s_i$ do calculate $cost^r(i) := \max_{i=1}^n \; d(r, s_i)$.
4. Select the string of minimum cost from the two previous steps.
```

Theorem 75.11

Let $1 \leq k < n$. *The algorithm* **Closest_Small_Substring** *is a* $(1 + \frac{1}{2k-1})$ *polynomial-time approximation algorithm for the CSSP when* $d_{min} \leq O(\log(nm))$.

Proof

To argue that **Closest_Small_Substring** takes polynomial time we notice by using Theorem 75.9 that $|P^T_{i_1,\ldots,i_k}| = O(k \log(nm))$ in step 1. This means that the inner loop of step 1 takes $|\Sigma|^{O(k \log(nm))} mnl = O((nm)^{O(\log|\Sigma|k)})$ time and that step 1 takes $O((nm)^{O(\log|\Sigma|k)})O((nm)^k) = O((nm)^{O(\log|\Sigma|k)})$ time. It is clear that steps 2 and 3 take less time.

Applying Theorems 75.9, 75.10 in a similar way to the one in the proof of Theorem 75.7, we get a $(1 + \frac{1}{2k-1})$ approximate solution. □

75.3.2.2 A $(1 + \epsilon)$ Polynomial-Approximation Scheme

The following ideas were first described in Ref. [40] and later in Ref. [35].

We would like to extend the algorithm **Closest_Small_Substring** for the general case. Let us first note that by trying all possibilities we get a set of indices $0 \le i_1 \le \cdots i_k \le n$ and substrings $T = \{t_{i_1}, \ldots, t_{i_k}\}$, which satisfy the conditions of Theorem 75.10 and hence we know that if we if we set $s'|_{Q^T_{i_1,\ldots,i_k}} = s_{i_1}|_{Q^T_{i_1,\ldots,i_k}}$ we get a "good" solution when we restrict ourselves to the positions $Q^T_{i_1,\ldots,i_k}$.

To calculate $s'|_{P^T_{i_1,\ldots,i_k}}$ we would like to use Theorem 75.5 applied to the strings $\tilde{t}_1, \ldots, \tilde{t}_n$ where $\tilde{t}_j = \bar{t}_j|_{P^T_{i_1,\ldots,i_k}}$. One major difficulty in doing so, is that we do not know $\bar{t}_1, \ldots, t_n$.

To do this we want for $t_j \lhd_l s_j$ to estimate $d(t_j, \bar{s})$. To accomplish that, we fix a small set $R \subset P^T_{i_1,\ldots,i_k}$ and we calculate $\bar{s}|_R$ by brute force. Based on the values of $\bar{s}|_R$ and $t_j|_R$, we define $f^R(t_j) \approx d(t_j, \bar{s})$. By choosing t'_j such that $f^R(t'_j) = \min_{t \lhd_l s_j} f^R(t)$ we hope to get a good approximation of $\bar{t}_j$ for $j = 1, \ldots, n$.

To make the preceding discussion more formal we introduce the following definitions and theorem.

Definition 75.14

Let $1 \le i_1 \le \cdots \le i_k \le n$, $T := \{\bar{t}_{i_1}, \ldots, \bar{t}_{i_k}\}$, *R be a multiset of positions from* $P^T_{i_1,\ldots,i_k}$ *and let us call* $\rho := \frac{|P^T_{i_1,\ldots,i_k}|}{|R|}$.

1. *For t a string of length l we define*

$$f^R(t) = \rho d(t|_R, \bar{s}|_R) + d\big(t\big|_{Q^T_{i_1,\ldots,i_k}}, \bar{t}_{i_1}\big|_{Q^T_{i_1,\ldots,i_k}}\big)$$

2. *For* $j = 1, \ldots, n$ *let* t'_j *be such that*

$$f^R(t'_j) := \min_{t \lhd_l s_j} f^R(t)$$

The following lemma implies that the t'_i referred to in Definition 75.14 is a "good" approximation of the $\bar{t}_i$, for $i = 1, \ldots, n$.

Theorem 75.12

Suppose the conditions of Definition 75.14 are met, and let s^* *be a string with* $s^*|_{P^T_{i_1,\ldots,i_k}} = \bar{s}|_{P^T_{i_1,\ldots,i_k}}$ *and* $s^*|_{Q^T_{i_1,\ldots,i_k}} = \bar{t}_{i_1}|_{Q^T_{i_1,\ldots,i_k}}$. *Furthermore, let us assume* $|R| = \frac{4}{\epsilon^2} \log nm$. *Then,*

$$\Pr\left(\left\{\forall i = 1, \ldots, n : d(s^*, t'_i) \le d(s^*, \bar{t}_i) + 2\epsilon\left|P^T_{i_1,\ldots,i_k}\right|\right\}\right) \le 2(nm)^{-\frac{1}{3}}$$

Proof

Let $P := P^T_{i_1,\ldots,i_k}$ and $Q := P^T_{i_1,\ldots,i_k}$. Let us choose $t \lhd_l s_i$ such that $d(s^*, t) \ge d(s^*, \bar{t}_i) + 2\epsilon|P|$.

Assume that $f(t) \leq f(\bar{t}_i)$ and consider the following cases:

- If $f(\bar{t}_i) < d(s^*, \bar{t}_i) + \epsilon|P|$ then

$$f(t) \leq d(s^*, \bar{t}_i) + \epsilon|P| \leq d(s^*, t) - 2\epsilon|P| + \epsilon|P| = d(s^*, t) - \epsilon|P|$$

- Similarly If $f(t) \leq f(\bar{t}_i)$ and $f(t) > d(s^*, t) - \epsilon|P|$, then

$$f(\bar{t}_i) \geq d(s^*, \bar{t}_i) - \epsilon|P| \geq d(s^*, \bar{t}_i) + 2\epsilon|P| - \epsilon|P| = d(s^*, \bar{t}_i) + \epsilon|P|$$

So if $f(t) \leq f(\bar{t}_i)$ we either have $f(t) \leq d(s^*, t) - \epsilon|P|$ or $f(\bar{t}_i) \geq d(s^*, \bar{t}_i) + \epsilon|P|$ and therefore,

$$\Pr(f(t) \leq f(\bar{t}_i)) \leq \Pr(f(t) \leq d(s^*, t_i) + \epsilon|P|) + \Pr(f(\bar{t}_i) \geq d(s^*, \bar{t}_i) + \epsilon|P|)$$

By using Lemma 75.1, it is clear that $d(s^*|_R, t|_R)$ is the sum of $|R| = \frac{4}{\epsilon^2} \log nm$ Bernoulli trials, each one indicating whether s^* and t coincide at the ith character in R. It is also clear that $\mu := E[d(s^*|_R, t|_R)] = \frac{d(s^*|_P, t|_P)}{\rho}$. Hence

$$\begin{aligned}
&\Pr(f(t) \leq d(s^*, t) - \epsilon|P|) \\
&= \Pr(\rho d(s^*|_R, t|_R) + d(s^*|_Q, t|_Q) \leq d(s^*, t) - \epsilon|P|) \\
&= \Pr(\rho d(s^*|_R, t|_R) \leq d(s^*|_P, t|_P) - \epsilon|P|) \\
&\leq \Pr(d(s^*|_R, t|_R) \leq \mu - \epsilon|R|) \leq e^{-\frac{1}{2}\epsilon^2|R|} \leq (nm)^{-2}
\end{aligned}$$

The previous to the last inequality follows from Lemma 75.3.
Similarly, we can prove that

$$\Pr(f(\bar{t}_i) \geq d(s^*, \bar{t}_i + \epsilon|P|) \leq (nm)^{-\frac{4}{3}}$$

In conclusion, for $t \lhd_l s_i$ such that $d(s^*, t) \geq d(s^*, \bar{t}_i) + 2\epsilon|P|$:

$$\Pr(f(t) \leq f(\bar{t}_i)) \leq 2(nm)^{-\frac{4}{3}}$$

Furthermore, we have

$$\begin{aligned}
\Pr(d(s^*, t'_i) \leq d(s^*, \bar{t}_i) + 2\epsilon|P|) &= \Pr\left(\bigcup_{t \lhd_l s_i} \{d(s^*, t) \leq d(s^*, \bar{t}_i) + 2\epsilon|P|\}\right) \\
&\leq \sum_{t \lhd_l s_i} \Pr(d(s^*, t) \leq d(s^*, \bar{t}_i) + 2\epsilon|P|) \leq m2(nm)^{-\frac{4}{3}} = 2n^{-\frac{4}{3}}m^{-\frac{1}{3}}
\end{aligned}$$

Hence, we know that

$$\begin{aligned}
&\Pr(\{\forall i = 1, \ldots, n : d(s^*, t'_i) \leq d(s^*, \bar{t}_i) + 2\epsilon|P|)\}) \\
&\leq \sum_{i=1}^{n} \Pr(d(s^*, t'_i) \leq d(s^*, \bar{t}_i) + 2\epsilon|P|) \leq n2n^{-\frac{4}{3}}m^{-\frac{1}{3}} < 2(nm)^{-\frac{1}{3}}
\end{aligned}$$

□

Algorithm Closest_Substring

```
1. for every set of substrings $T = \{t_{i_1}, \ldots, t_{i_k}\}$ where $t_{i_j} \lhd_l s_{i_j}$ $j = 1, \ldots, k$ do
    (a) Let $R$ be a multiset containing $\frac{4}{\epsilon^2} \log nm$ uniformly random positions from $P^T_{i_1,\ldots,i_k}$.
    (b) By enumerating all strings in $\Sigma^R$ find $\bar{s}|_R$.
    (c) for $j = 1, \ldots, n$ do find $t'_j \lhd_l s_j$ that satisfies Definition 75.14.
    (d) By using Theorem 75.5 find a solution $s'$ to the closest string problem for
        $t'_1|_{P^T_{i_1,\ldots,i_k}}, \ldots, t'_n|_{P^T_{i_1,\ldots,i_k}}$.
    (e) Define $x$ so that $x|_{Q_{i_1,\ldots,i_k}} = s_{i_1}|_{Q^T_{i_1,\ldots,i_k}}$ and $x|_{P^T_{i_1,\ldots,i_k}} = s'$.
    (f) Call $cost^T(i_1, \ldots, i_k) := \max_{i=1}^{n} d(x, s_i)$.
2. Let $x'$ be the string that minimizes $cost^T(i_1, \ldots, i_k)$
```

3. **for i=:1**, $\dots$, n and **for** every $r \triangleleft_l s_i$ **do** calculate $cost^r(i) := \max_{i=1}^{n} \; d(r, s_i)$.
4. Select the string with the minimum cost from the two previous steps.

Theorem 75.13

Let $0 < \delta < 1$. *The algorithm* **Closest_Substring** *is a* $(1 + \delta)$ *polynomial-approximation algorithm for the CSP.*

Proof

Let s' be the output of algorithm **Closest_Substring**. Using Theorem 75.12 we note that

$$d(s^*, t_i') \le d(s^*, \bar{t}_i) + 2\epsilon |P^T_{i_1,\dots,i_k}| \text{ with high probability}$$

An application of Theorem 75.9 implies that $|P^T_{i_1,\dots,i_k}| \le k\bar{d}_{min}$. Combining this with Theorem 75.10,

$$d(s^*, t_i') \le \left(1 + \frac{1}{2k-1} + 2\epsilon k\right) \bar{d}_{min} \text{ for } i = 1, \dots, n \text{ with high probability}$$

Employing Theorem 75.5 with $\epsilon' = \epsilon |P^T_{i_1,\dots,i_k}| \le \epsilon k \bar{d}_{min}$,

$$d(s', t_i') \le \left(1 + \frac{1}{2k-1} + 3\epsilon k\right) \bar{d}_{min} \text{ for } i = 1, \dots, n, \text{ with high probability}$$

Then if we choose δ such that $\left(\frac{1}{2k-1} + 3\epsilon k\right) \le \delta$ the result holds. □

By using the method of conditional probabilities [38] it is possible to eliminate the randomness in algorithm **Closest_Substring**. More details can be found in Ref. [35].

75.4 Conclusions

In this chapter we have considered the problem of selecting primers for a given set of DNA sequences that will be employed in PCR and MP-PCR experiments. The basic version of the problem, namely, the PSP and its variant, the DPSP, were discussed in detail. Some of the salient algorithms found in the literature for these versions of primer selection were surveyed. An elaborate discussion of other variants of degenerate primer design can be found in Ref. 1. We have also addressed the PMS problem and some of the algorithms to solve this problem. Furthermore we consider the related minimization problems: the closest string (CSP), and the closest substring problems (CSSP). For the CSP and the CSSP we presented a survey of the existing polynomial approximation algorithms.

Acknowledgments

This work has been supported in part by the NSF Grant ITR-0326155.

References

[1] Linhart, C. and Shamir, R., The degenerate primer design problem—theory and applications, *J. Comput. Biol.*, 12(4), 431, 2005.

[2] Pearson, W. R., Robins, G., Wrege, D. E., and Zhang, T., On the primer selection problem in polymerase chain reaction experiments, *Disc. Appl. Math.*, 71, 231, 1996.

[3] Chamberlain, J. S., Gibbs, R. A., Rainer, J. E., Nguyen, P. N., and Casey, C. T., Deletion screening of the Duchenne muscular dystrophy locus via multiplex DNA amplification, *Nucl. Acids Res.*, 16, 11141, 1988.

[4] Kwok, S., Chang, S. Y., Sninsky, J. J., and Wang, A., A guide to the design and use of mismatched and degenerate primers, *PCR Methods Appl.*, 3, 39, 1994.

[5] Souvenir, R., Buhler, J., Stormo, G., and Zhang, W., Selecting degenerate multiplex PCR primers, *Proc. Int. Workshop on Algorithms in Bioinformatics*, 2003, p. 512.

[6] Doi, K. and Imai, H., Greedy algorithms for finding a small set of primers satisfying cover length resolution conditions in PCR experiments, *Proc. Workshop on Genome Informatics*, 1997, p. 43.

[7] Doi, K. and Imai, H., A Greedy algorithm for minimizing the number of primers in multiple PCR experiments, *Genome Informatics*, 10, 1999, pp. 73–82.

[8] Doi, K. and Imai, H., Complexity properties of the primer selection problem for PCR experiments, *Proc. Japan–Korea Joint Workshop on Algorithms and Computation*, 2000, p. 152.

[9] Konwar, K. M., Mandoiu, I. I., Russell, A. C., and Shvartsman, A. A., Improved algorithms for multiplex PCR primer set selection with amplification length constraints, *Proc. of the 3rd Asia-Pacific Bioinformatics Conference (APBC)*, Singapore, 2005, pp. 41–50.

[10] Rose, T. M., Schultz, E. R., Henikoff, J. G., Pietrokovski, S., McCallum, C. M., and Henikoff, S., Consensus-degenerate hybrid oligonucleotide primers for amplification of distantly related sequences, *Nucleic Acids Res.*, 26(7), 1628, 1998.

[11] Fuchs, T., Malecova, B., Linhart, C., Sharan, R., Khen, M., Herwig, R., Shmulevich, D., Elkon, R., Steinfath, M., O'Brien, J. K., Radelof, U., Lehrach, H., Lancet D., and Shamir, R., DEFOG: A Practical Scheme for Deciphering Families of Genes, *Genomics*, 80(3), 2002, pp. 1–8.

[12] Wei, X., Kuhn, D. N., and Narasimhan, G., Degenerate primer design via clustering, *Proc. Bioinformatics Conf.*, 2003, p. 75.

[13] Balla, S., Rajasekaran, S., and Mandoiu, I. I., Efficient Algorithms for Degenerate Primer Search, UConn BECAT/CSE Technical report, BECAT/CSE-TR-05-2, 2005.

[14] Buhler, J. and Tompa, M., Finding motifs using random projections, *Proc. of the fifth Annual International Conference on Computational Molecular Biology (RECOMB)*, Montreal, Canada, 2001, pp. 69–76.

[15] Rajasekaran, S., Balla, S., and Huang, C.-H., Exact algorithms for planted motif problems, *Journal of Computational Biology*, 12(8), 2005, pp. 1117–1128.

[16] Keich, U. and Pevzner, P., Finding motifs in the twilight zone, *Bioinformatics*, 18, 1374, 2002.

[17] Eskin, E. and Pevzner, P., Finding composite regulatory patterns in DNA sequences, *Bioinformatics*, 18, 354, 2002.

[18] Price, A., Ramabhadran S., and Pevzner, P. A., Finding subtle motifs by branching from sample strings, *Bioinformatics*, 19(1), 1, 2003.

[19] Hertz, G. and Stormo, G., Identifying DNA and protein patterns with statistically significant alignments of multiple sequences, *Bioinformatics*, 15, 563, 1999.

[20] Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., and Wootton, J. C., Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment, *Science*, 262, 208, 1993.

[21] Bailey, T. L. and Elkan, C., Fitting a mixture model by expectation maximization to discover motifs in biopolymers, *Proc. Int. Conf. on Intelligent Systems for Molecular Biology*, 1994, p. 28.

[22] Martinez, H. M., An efficient method for finding repeats in molecular sequences, *Nucleic Acids Res.*, 11(13), 4629, 1983.

[23] Brazma, A., Jonassen, I., Vilo, J., and E. Ukkonen, E., Predicting gene regulatory elements in silico on a genomic scale, *Genome Res.*, 15, 1202, 1998.

[24] Galas, D. J., Eggert, M., and Waterman, M. S., Rigorous pattern-recognition methods for DNA sequences: analysis of promoter sequences from *Escherichia coli*, *J. Mole. Biol.*, 186(1), 117, 1985.

[25] Sinha, S. and Tompa, M., A statistical method for finding transcription factor binding sites, *Proc. Int. Conf. on Intelligent Systems for Molecular Biology*, 2000, p. 344.

[26] Staden, R., Methods for discovering novel motifs in nucleic acid sequences, *Comput. Appli. Biosciences*, 5(4), 293, 1989.

[27] Tompa, M., An exact method for finding short motifs in sequences, with application to the ribosome binding site problem, *Proc. Int. Conf. on Intelligent Systems for Molecular Biology*, 1999, p. 262.

[28] van Helden, J., Andre, B., and Collado-Vides, J., Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies, *J. Mole. Biol.*, 281(5), 827, 1998.

[29] Pevzner, P. and Sze, S.-H., Combinatorial approaches to finding subtle signals in DNA sequences, *Proc. Int. Conf. on Intelligent Systems for Molecular Biology*, 2000, p. 269.

[30] Lawrence, C. E. and Reilly, A. A., An Expectation Maximization (EM) Algorithm for the identification and characterization of common sites in unaligned biopolymer sequences, *Proteins: Struct., Function, Genet.*, 7, 41, 1990.

[31] Rajasekaran, S., Motif search algorithms, in *Handbook of Computational Molecular Biology*, CRC Press, Boca Raton, FL, 2005.

[32] Frances, M. and Litman, A., On covering problems of codes, *Theor. Comput. Syst.*, 30, 113, 1997.

[33] Ben-Dor, A., Lancia, G., Perone, J., and Ravi, R., Banishing bias from consensus sequences, *Proc. Symp. on Combinatorial Pattern Matching Theory Comput. Systems*, Lecture Notes in Computer Science, Vol. 1264, Springer, Berlin, 1997, p. 247.

[34] Li, M., Ma, B., and Wang, L., Finding similar regions in many strings, *Proc. STOC*, 1999, p. 473.

[35] Li, M., Ma, B., and Wang, L., On the closest string and substring problems, *JACM*, 49(2), 157, 2002.

[36] Gramm, J., Niedermeier, R., and Rossmanith, P., Exact solutions for closest string and related problems, *Proc. Symp. on Algorithms and Computation*, Lecture Notes in Computer Science, Vol. 2223, Springer, Berlin, 2001, p. 441.

[37] Gramm, J., Niedermeier, R., and Rossmanith, P., Fixed-parameter algorithms for closest string and related problems, *Algorithmica*, 37(1), 25, 2003.

[38] Motwani, R. and Raghavan, P., *Randomized Algorithms*, Cambridge University Press, London, 1995.

[39] Lanctot, K., Li, M. Ma, B., Wang, S., and Zhang, L., Distinguishing string selection problems, *Proc. SODA*, 1999, p. 633.

[40] Ma, B., A polynomial time approximation scheme for the closest substring problem, *Proc. Combinatorial Pattern Matching Symp.*, Lecture Notes in Computer Science, Vol. 1848, Springer, Berlin, 2000, p. 99.

[41] Linhart, C. and Shamir, R., The degenerate primer design problem, *Bioinformatics*, 18(1), 172, 2002.

76

Dynamic and Fractional Programming-Based Approximation Algorithms for Sequence Alignment with Constraints

Abdullah N. Arslan
University of Vermont

Ömer Eğecioğlu
University of California, Santa Barbara

76.1 Introduction

There are interesting algorithmic issues that arise when length constraints are taken into account in the formulation of a variety of problems on string similarity, particularly in the problems related to local alignment. These types of problems have their roots and most striking applications in computational biology. In fact, because of the applications in biological sequence analysis, detection of local similarities in two given strings has become an increasingly important computational problem. When there are additional constraints that need to be satisfied as a part of the search criteria, it is natural to consider approximation algorithms for the resulting computational problems for large parameters.

Given two strings X and Y, the classical dynamic programming solution to the local alignment problem searches for two substrings $I \subseteq X$ and $J \subseteq Y$ with maximum similarity score under a given scoring scheme, where $\subseteq$ indicates the substring relation. This classical definition of similarity has certain anomalies mainly because the lengths of the segments I and J are not taken into account. To cope with the possible anomalies of mosaic and shadow effects, many variations of the local alignment problem have been suggested. Mosaic effect is observed when an unrelated segment is sandwiched between two very similar segments. Shadow effect is observed when a biologically important short alignment is not detected because it overlaps with a longer yet biologically inadequate alignment with only a slightly higher score.

The variations suggested either define new objective functions, or include a length constraint on the substrings I and J for optimal alignments sought. This constraint can be driven by practical considerations for various objective functions (e.g., the maximization of *length-normalized* scores) and can be explicitly given such as requiring $|I| + |J| \geq t$ or $|J| \leq T$ for given parameters t and T. In addition, in some local alignment problems the constraint may also be implicit, as it happens in the case of cyclic sequence comparison. In Table 76.1 we give a list of local alignment problems, their objectives, and computational results for them. The function $s(I, J)$ denotes the similarity score between I and J. The optimizations are over all possible substrings I of X, and J of Y. In the table, we use "nor. score" as a shorthand for length-normalized score. For any optimization problem $\mathcal{P}$, we denote by $\mathcal{P}^*$ its optimum value, and sometimes drop the parameters from the notation when they are obvious from the context. An optimization problem $\mathcal{P}$ is called *feasible* if it has a solution with the given parameters.

In most cases under consideration, there are simple dynamic programming formulations for the solution of the exact version of a given alignment problem with a length constraint. However, the resulting algorithms require cubic or higher time complexity, which is unacceptably high for practical purposes since the sequence lengths can be on the order of millions. To cope with such high complexity, approximations are considered both in definitions of similarity, and in the resulting computations.

There have been approximation algorithms proposed for various alignment problems with constraints, involving applications of techniques from fractional programming, and dynamic programming. In this chapter, we present a survey of the most interesting approximation algorithms for variations of local alignment problems. Our focus is on fractional programming algorithms, and algorithms returning results that meet the length constraint only partially but guaranteed to be within a given tolerance. These algorithms can be organized into three main categories:

1. *Fractional programming algorithms.* Application of fractional programming on *adjusted normalized local alignment* (the *ANLA* problem in Table 76.1) is of interest. The local alignment is normally defined as a graph problem. The fractional programming technique offers an iterative solution such that at each iteration an ordinary local alignment problem with modified weights is solved. This mimics the action of manually changing the weights until the results are found satisfactory. Fundamental theorems of fractional programming guarantee an optimal solution at the conclusion of these iterations. The termination properties of the iterative scheme are not obvious at all without referring to the results established for fractional programming.
2. *Approximation algorithms for partial constraint satisfaction.* Another noteworthy feature of some constrained local alignment approximation algorithms is their unusual performance measure. Ordinarily, performance of an approximation algorithm is measured by comparing the returned results against optimum value with respect to the objective function. In some approximation results regarding the length-constrained local alignment problems, such as the problem of finding a sufficiently long alignment with high score (the *LAt* problem in Table 76.1), the alignment returned is assured to have at least the score obtainable with respect to the given constraint, but the length constraint is satisfied to only within a prescribed tolerance from the required length value.
3. *Fractional programming approximation algorithms.* There are fractional programming approximation algorithms for the *normalized local alignment* problem with length constraint (the *NLAt* problem in Table 76.1). These algorithms iteratively invoke an approximation algorithm to solve a length-constrained local alignment problem (*LAt*) such that the length constrained is guaranteed to be satisfied within a given tolerance. This length guarantee carries over for the final result for the normalized local alignment problem. That is, the fractional programming algorithm returns an approximate result for which the guarantee on the satisfaction of the length constraint within some tolerance is due to the approximation algorithm used at each iteration, and the criteria is preserved over the iterations.

In this chapter, we start with the basic framework for local alignment in Section 76.2. We present the details of the topics enumerated above in three sections. In Section 76.3 we describe the fractional programming algorithms for the adjusted normalized local alignment problem (*ANLA*). In Section 76.4 we describe an approximation algorithm that uses decomposition of the alignment graph into slabs in order to find a sufficiently long alignment with high score (*LAt*). This algorithm is used to obtain a fractional programming approximation algorithm for the normalized local alignment problem (*NLAt*). This is done in such a way that the length constraint is met within a given tolerance, as described in detail in Section 76.5. The main results and the algorithm descriptions given in the subsequent sections of this chapter are a compilation and a reorganization of the results that appear in Refs. [1–4].

76.2 Framework for Pairwise Sequence Comparison

Given two strings $X = x_1x_2 \dots x_n$ and $Y = y_1y_2 \dots y_m$ with $n \geq m$, we use the *alignment graph* $G_{X,Y}$ to analyze *alignments* between all substrings of X and Y. The alignment graph is a directed acyclic graph having $(n + 1)(m + 1)$ lattice points (u, v) as vertices for $0 \leq u \leq n$ and $0 \leq v \leq m$. Figure 76.1 shows an alignment graph for $x_i \cdots x_k = ATTGT$ and $y_j \cdots y_l = AGGACAT$. Matching diagonal arcs are drawn as solid lines while mismatching diagonal arcs are shown by dashed lines. Dotted lines are used for horizontal and vertical arcs. An example alignment path is shown in Figure 76.1. Labels of the arcs on this path are the corresponding edit operations where ϵ denotes the null string. An *alignment path* for substrings $x_i \cdots x_k$ and $y_j \cdots y_l$ is a directed path from the vertex $(i - 1, j - 1)$ to (k, l) in $G_{X,Y}$ where $i \leq k$ and $j \leq l$. To each vertex there is an incoming arc from each neighbor if it exists. Horizontal and vertical arcs correspond to insert and delete operations, respectively. We sometimes use *indel* to refer to an insert or a delete operation. The diagonal arcs correspond to substitutions which are either matching (if the corresponding symbols are the same) or mismatching (otherwise). If we trace the arcs of an alignment path for substrings I and J and perform the indicated edit operations in the given order on I, we obtain J .

Blocks of insertions and deletions are also referred to as *gaps*. The alignment in Figure 76.1 includes two gaps with sizes 1 and 3. We will use the terms alignment and alignment path interchangeably.

The objective of sequence alignment is to quantify the similarity between X and Y under a given *scoring scheme*. In the *simple scoring scheme*, the arcs of $G_{X,Y}$ are assigned weights determined by nonnegative reals δ (*mismatch penalty*) and μ (*indel* or *gap penalty*). We assume that $s(x_i, y_j)$ is the similarity score between the symbols x_i and y_j which is normally 1 for a match $(x_i = y_j)$ and $-\delta$ for a mismatch $(x_i \neq y_j)$.

Given two strings X and Y the *local alignment* (*LA*) problem seeks substrings $I \subseteq X$ and $J \subseteq Y$ with the highest similarity score. The optimum value $LA^*(X, Y)$ for this problem is given by

$$LA^*(X, Y) = \max\{s(I, J) \mid I \subseteq X, J \subseteq Y\} \tag{76.1}$$

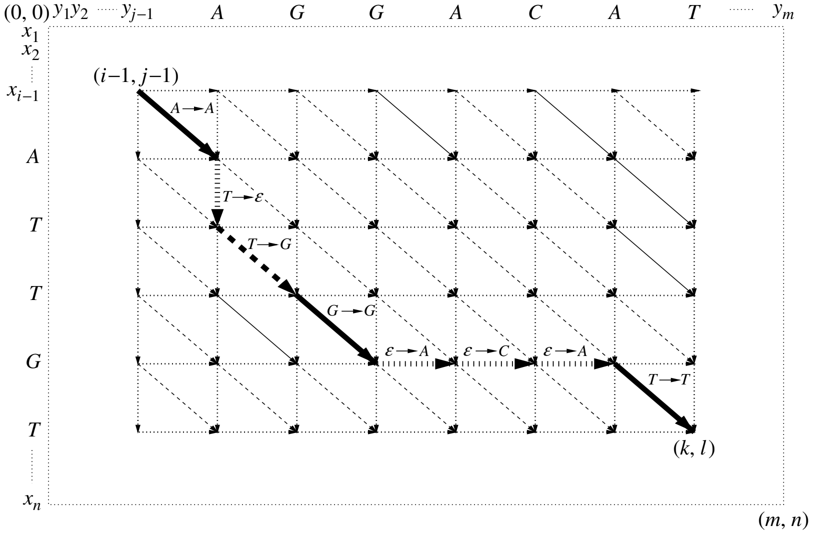


FIGURE 76.1 Alignment graph $G_{X,Y}$ where $x_i \cdots x_k = ATTGT$ and $y_j \cdots y_l = AGGACAT$.

where $s(I, J)$ is the best alignment score between I and J. Alignments have positive scores, or otherwise, they do not exist, that is, $s(I, J) = 0$ iff there is no alignment between I and J.

The following is the classical dynamic programming formulation [5] to compute the maximum local alignment score $\mathcal{S}_{i,j}$ achieved by an optimal local alignment ending at each vertex (i, j):

$$\mathcal{S}_{i,j} = \max\{0,\ \mathcal{S}_{i-1,j} - \mu,\ \mathcal{S}_{i-1,j-1} + s(x_i, y_j),\ \mathcal{S}_{i,j-1} - \mu\} \tag{76.2}$$

for $1 \leq i \leq n$, $1 \leq j \leq m$, with the boundary conditions $\mathcal{S}_{i,j} = 0$ whenever $i = 0$ or $j = 0$. Then

$$LA^*(X, Y) = \max_{i,j} \mathcal{S}_{i,j} \tag{76.3}$$

LA^* can be computed using the Smith–Waterman algorithm [6] in time $O(nm)$. The space complexity is $O(m)$ because only $O(m)$ entries of the dynamic programming matrix need to be stored at any given time.

The simple scoring scheme can be extended such that the scores can vary depending on the individual symbols within the same edit operation type. This leads to arbitrary scoring matrices. In this case there is a dynamic programming formulation similar to Eq. (76.2).

Affine gap penalties is another common scoring scheme in which the total penalty for a gap of size k, that is, a block of k insertions (or deletions), is $\alpha + (k-1)\mu$ where α is the gap open penalty, and μ is called the gap extension penalty. The dynamic programming formulation for this case can be described as follows (see Ref. [5]): Let $\mathcal{E}_{i,j} = \mathcal{F}_{i,j} = \mathcal{S}_{i,j} = 0$ when i or j is 0, and define

$$\begin{aligned}\mathcal{E}_{i,j} &= \max\{\mathcal{S}_{i,j-1} - \alpha,\ \mathcal{E}_{i,j-1} - \mu\}\\ \mathcal{F}_{i,j} &= \max\{\mathcal{S}_{i-1,j} - \alpha,\ \mathcal{F}_{i-1,j} - \mu\}\\ \mathcal{S}_{i,j} &= \max\{0,\ \mathcal{S}_{i-1,j-1} + s(x_i, y_j),\ \mathcal{E}_{i,j},\ \mathcal{F}_{i,j}\}\end{aligned} \tag{76.4}$$

By virtue of this formulation, consideration of affine gap penalties does not increase the asymptotic complexity of the local alignment problem.

We can also express the alignment problems as optimization problems that involve linear functions. In the following sections we will describe fractional programming algorithms based on these expressions. We define an *alignment vector* as the vector of edit operation frequencies such that the scores and the lengths of alignments can be expressed as linear functions over alignment vectors. For example, under the basic scoring scheme, we say that (x, y, z) is an alignment vector if there is an alignment path between substrings $I \subseteq X$ and $J \subseteq Y$ with x matches, y mismatches, and z indels. In Figure 76.1, (3, 1, 4) is an alignment vector corresponding to the path shown in the figure. Let AV, under a given scoring scheme, denote the set of alignment vectors. Then $s(I, J)$ can be expressed as a linear function $SCORE$ over AV for the scoring schemes we study: the basic scoring scheme, arbitrary scoring matrices, and affine gap penalties. For example when simple scoring is used

$$SCORE(a) = x - \delta y - \mu z \quad \text{for} \quad a = (x, y, z) \in AV$$

where x,y,z of alignment vector a represent the number of matches, mismatches, and indels, respectively. We can easily verify that also for affine gap penalties and arbitrary scoring matrices, $SCORE$ can be expressed as a linear function.

The local alignment problem LA can be rewritten as follows:

$$LA : \text{maximize } SCORE(a) \quad \text{s.t. } a \in AV$$

76.3 Fractional Programming *ANLA* Algorithms

Using length-normalized scores in local alignment is suggested by Arslan et al. [2] to cope with the mosaic and shadow effects. The objective of the *NLAt* problem [2] is

$$NLAt^*(X, Y) = \max\{s(I, J)/(|I| + |J|) \mid I \subseteq X, J \subseteq Y, |I| + |J| \geq t\} \tag{76.5}$$

To solve the *NLAt* problem we can extend the dynamic programming formulation for the scoring schemes that we address in this chapter by adding another dimension. At each entry of the dynamic programming matrix we can store optimum scores for all possible alignment lengths up to $m + n$. This increases the time and space complexities to $\Theta(n^2m)$ and $\Theta(nm)$, respectively. These are unacceptably high because in practice the values of both n and m may be on the order of millions.

The length of an alignment can appropriately be defined as the sum of the lengths of the substrings involved in the alignment. For an alignment vector $a \in AV$, the length of the corresponding alignment can be expressed as a linear function *LENGTH*. For example, when the simple scoring scheme is used

$$LENGTH(a) = 2x + 2y + z \quad \text{for} \quad a = (x, y, z) \in AV$$

where x,y,z represent the number of matches, mismatches, and indels, respectively. We can easily see that for affine gap penalties and arbitrary scoring matrices *LENGTH* can be expressed as a linear function. We assume that only the matches have nonnegative scores; therefore on any alignment the score cannot exceed the length.

The objective of *NLAt* may be achieved by a reformulation. In *adjusted normalized local alignment* (*ANLA*) problem, we can modify the maximization ratio function in such a way that we drop the length constraint, yet achieve a similar objective: to obtain sufficiently long alignments with a high degree of similarity. The adjusted length-normalized score of an alignment is computed by adding some parameter $L \geq 0$ to the denominator in the calculation of the quotient of ordinary scores by the length. Thus the *ANLA* problem [2] is a variant of the normalized local alignment problem in which the length constraint is dropped, and the optimization function is modified by adding a parameter L to the denominator.

$$ANLA^*(X, Y) = \max\{s(I, J)/(|I| + |J| + L) \mid I \subseteq X, J \subseteq Y\} \tag{76.6}$$

The adjusted normalized local alignment problem *ANLA* can be rewritten as follows:

$$ANLA : \text{maximize } \frac{SCORE(a)}{LENGTH(a)+L} \quad \text{s.t. } a \in AV$$

For *ANLA* faster algorithms are possible using *fractional programming* technique. The provable time complexity of the *ANLA* problem for rational weights is $O(nm\log\, n)$, as we discuss later. Test results of a fractional programming-based approach suggest that the time complexity is $O(nm)$, although this result is empirical. Compared to $\Theta(n^2m)$ time complexity of a naive dynamic programming algorithm for the *NLAt* problem, the *ANLA* problem can be solved much faster.

Fractional programming *ANLA* algorithms [2] use the *parametric method.* They iteratively solve a so-called *parametric problem* LA_λ which is the following optimization problem: for a given λ

$$LA_\lambda^*(X, Y) = \max\{s(I, J) - \lambda(|I| + |J| + L) \mid I \subseteq X, J \subseteq Y\} \tag{76.7}$$

$LA_\lambda(X, Y)$ can also be written as

$$LA(\lambda) : \text{maximize } SCORE(a) - \lambda\ LENGTH(a) - \lambda L \quad \text{s.t. } a \in AV$$

Proposition 76.1 **(Arslan et al. [2])**

For $\lambda < \frac{1}{2}$, the optimum value $LA^*(\lambda)$ of the parametric LA problem can be formulated in terms of the optimum value LA^* of an LA problem.

Proof

Under the simple scoring scheme the optimum value of the parametric problem, when $\lambda < \frac{1}{2}$, is

$$LA_{\delta,\mu}^*(\lambda) = (1 - 2\lambda)LA_{\delta',\mu'}^* - \lambda L, \quad \text{where} \quad \delta' = \frac{\delta + 2\lambda}{1 - 2\lambda},\ \mu' = \frac{\mu + \lambda}{1 - 2\lambda} \tag{76.8}$$

We can easily verify that a similar relation exists in the case of arbitrary scoring matrices, and affine gap penalties. Thus, computing $LA^*(\lambda)$ involves solving the local alignment problem *LA*, and performing some simple arithmetic afterward. □

```
Algorithm Dinkelbach
Pick an arbitrary alignment, and let $\lambda^*$ be the adjusted length-normalized score of this alignment
Repeat
   $\lambda \leftarrow \lambda^*$
   Solve $LA(\lambda)$ and let $\lambda^*$ be the adjusted length-normalized score of an optimal alignment
Until $\lambda^* = \lambda$
Return($\lambda^*$)
```

FIGURE 76.2 Dinkelbach algorithm for *ANLA* [2].

We assume, without loss of generality, that for any alignment the score does not exceed the number of matches. Therefore for any alignment, its normalized score $\lambda \leq \frac{1}{2}$. We consider $\lambda = \frac{1}{2}$ as a special case since it can only happen when the alignment is composed of matches only, and $L = 0$.

The thesis of the parametric method of fractional programming is that the optimum solution to the original problem that involves a ratio of two functions can be obtained via optimal solutions of the parametric problem. In this case, an optimal solution to a ratio optimization problem *ANLA* can be achieved via a series of optimal solutions of the parametric problem $LA(\lambda)$ with different parameters λ. In fact $\lambda = ANLA^*$ iff $LA^*(\lambda) = 0$. That is, an alignment vector $v \in AV$ has the optimum adjusted normalized score λ iff v is an optimal alignment vector for the parametric problem $LA(\lambda)$ with optimum value zero. (See Ref. [2] for more details, also see Refs. [7,8] for many interesting properties of fractional programming). The *Dinkelbach* algorithm for the *ANLA* problem is shown in Figure 76.2. Solutions of the parametric problems through the iterations yield improved (higher) values to λ except for the last iteration in which λ remains the same, and becomes the optimum value. In fractional programming algorithms convergence to an optimal result is guaranteed: In infinite sets the convergence to optimum is super-linear. In finite sets the termination is guaranteed. In the case of *ANLA* $Dinkelbach$ algorithm, when the algorithm terminates, the final alignment is optimal with respect to both the ordinary scores used at that iteration, and the adjusted length-normalized scoring with the original scores. This mimics manually changing the scores until the result is satisfactory.

As reported by Arslan et al. [2], experiments suggest that the number of iterations in the algorithm is a small constant: 3–5 on average. However, a theoretical bound is yet to be established. If we assume that the sequences involved in alignments are fixed (e.g., consider the normalized global alignment), and the simple scoring scheme is used then the number of iterations is bounded by the size of the convex hull of lattice points whose diameter is bounded by the length of the strings. In this case, each parametric problem is optimized at one of the extreme points of the convex hull, and each extreme point is visited at most once during the iterations. It is known that the size of a convex hull of diameter N is $O(N^{2/3})$ (see, e.g., Ref. [1]). Even this rough estimate shows that the algorithm in the worst case is better than the straightforward dynamic programming extension for *ANLA*.

In practice the scores are rational, and in the case of rational scores there is a provably better result [2] which is achieved by Algorithm *RationalANLA* given in Figure 76.3. The algorithm uses Megiddo's technique [9] to perform a binary search for optimum adjusted normalized score over an interval of

```
Algorithm RationalANLA
Let $\sigma$ be the smallest gap between two adjusted length normalized scores
Initialize $[e, f] \leftarrow [0, \frac{1}{2}\sigma^{-1}]$
While ($e + 1 < f$) do
   $k \leftarrow \lfloor (e + f)/2 \rfloor$
   If $LA^*(k\sigma) > 0$ then $e \leftarrow k$ else $f \leftarrow k$
End {while}
Return($e\sigma$)
```

FIGURE 76.3 *ANLA* algorithm RationalANLA for rational scores [2].

integers. The search is based on the sign of the optimum value of the parametric problem. In this case, if $LA^*(\lambda) = 0$, then $\lambda = ANLA^*$, and an optimal alignment vector of $LA(\lambda)$ is also an optimal solution of *ANLA*. In contrast, if $LA^*(\lambda) > 0$, then a larger λ, and if $LA^*(\lambda) < 0$, then a smaller λ should be tested (i.e., Problem $LA(\lambda)$ should be solved with a different value of λ). When the scores are rational numbers the effective search space includes $O(n^2)$ integers because the gap between any two distinct length-normalized score is $\Omega(1/n^2)$. The algorithm solves $O(\log\ n)$ parametric problems. Therefore the resulting time complexity is $O(nm \log\ n)$, and the space complexity is $O(m)$.

76.4 Approximation Algorithms for Partial Constraint Satisfaction

In Table 76.1 we list several local alignment problems with length constraint. For these problems there are approximation algorithms that guarantee the satisfaction of the constraints partially, that is, they return alignments whose lengths are within a given tolerance of the required length.

These algorithms decompose the alignment graph into slabs. The length-restricted local alignment *LRLA* problem [3] is suggested to find alignments with optimal score over the alignments that involve substrings of up to a given length. The length limit is only on the substrings of one of the strings. The approximation algorithms for this problem imagine that the alignment graph is partitioned into vertical slabs. The results for this problem are summarized in Table 76.1. The cyclic sequence alignment *CLA* [3] is a special case of the *LRLA* problem. In the *CLA* problem the length constraint is implicit as shown in the table. The *LRLA* algorithms and results are applicable to the *CLA* problem, too.

We omit the details of *LRLA* approximation algorithms. Instead we describe another algorithm which is also based on the decomposition of the alignment graph into slabs. This algorithm is for the length-constrained local alignment problem *LAt* [4] (see also Table 76.1).

TABLE 76.1 Variations of Local Alignment Problems [4]

Alignment problem	Objective	Algorithm	Time	Space	Returned alignment satisfies
LA	maximize $s(I, J)$	Smith–Waterman	$O(nm)$	$O(m)$	Score $= LA^*$
ANLA	maximize $\frac{s(I,J)}{\lvert I\rvert+\lvert J\rvert+L}$ for parameter $L \geq 0$	Dinkelbach	$O(nm)$ (experimental)	$O(m)$	Score $= ANLA^*$
		Rational*ANLA*	$O(nm \log\ n)$	$O(m)$	Score $= ANLA^*$
LRLA	maximize $s(I, J)$ such	*HALF*	$O(nm)$	$O(m)$	Score $\geq \frac{1}{2} LRLA^*$
	that $\lvert J\rvert \leq T$	*APX-LRLA*	$O(nmT/\Delta)$	$O(mT/\Delta)$	Score $\geq LRLA^* - 2\Delta$
CLA	*LRLA* with parameters X, YY, and $T = \lvert Y\rvert$	The same *LRLA* algorithms, complexity, and results			
LAt	maximize $s(I, J)$ such that $\lvert I\rvert + \lvert J\rvert \geq t$	*APX-LAt*	$O(rnm)$	$O(rm)$	Score $\geq LAt^*$, length $\geq (1 - \frac{1}{r})t$
Qt	find (I, J) such that $\frac{s(I,J)}{\lvert I\rvert+\lvert J\rvert} > \lambda$, and $\lvert I\rvert + \lvert J\rvert \geq t$, for parameter $\lambda > 0$	*APX-LAt*	$O(rnm)$	$O(rm)$	Nor. score $> \lambda$, length $\geq (1 - \frac{1}{r})^t$
NLAt	maximize $\frac{s(I,J)}{\lvert I\rvert+\lvert J\rvert}$ such that $\lvert I\rvert + \lvert J\rvert \geq t$	Dinkelbach	$O(rnm)$ (experimental)	$O(rm)$	Nor. score $\geq NLAt^*$, length $\geq (1 - \frac{1}{r})t$
		Rational*NLAt*	$O(rnm \log n)$	$O(rm)$	Nor. score $\geq NLAt^*$, length $\geq (1 - \frac{1}{r})t$

For a given t, we define the *local alignment with length threshold* score between X and Y as

$$LAt^*(X, Y) = \max\{s(I, J) \mid I \subseteq X, J \subseteq Y, \text{ and } |I| + |J| \geq t\} \tag{76.9}$$

Equivalently

$$LAt : \text{maximize } SCORE(a) \quad \text{s.t. } a \in AVt$$

where $AVt \subseteq AV$ is the set of alignment vectors corresponding to alignments with length $\geq t$.

Although the problem itself is not very interesting, an algorithm for the problem can be used to find a long alignment with length-normalized score $> \lambda$ for a given positive λ, which is a practical query problem Qt included in Table 76.1. We also show that the algorithm for the local alignment with length threshold leads to improved approximation algorithms for the normalized local alignment problem (see Section 76.5).

To solve LAt we can extend the dynamic programming formulation in Eq. (76.2) by adding another dimension. At each entry of the dynamic programming matrix we store optimum scores for all possible lengths up to $m + n$, increasing the time and space complexities to $\Theta(n^2 m)$ and $\Theta(nm)$, respectively.

We describe an approximation algorithm *APX-LAt* [4] which computes a local alignment whose score is at least LAt^*, and whose length is at least $(1 - \frac{1}{r})t$ provided that the LAt problem is feasible, that is the algorithm finds two substrings $\widehat{I} \subseteq X$, and $\widehat{J} \subseteq Y$ such that $s(\widehat{I}, \widehat{J}) \geq LAt^*$ and $|\widehat{I}| + |\widehat{J}| \geq (1 - \frac{1}{r})t$. The algorithm runs in time $O(rnm)$ using $O(rm)$ space. For simplicity, we assume the simple scoring scheme. Instead of a single score, we maintain at each node (i, j) of $G_{X,Y}$, a list of alignments with the property that for positive s where s is the optimum score achievable over the set of alignments with length $\geq t$ and ending at (i, j), at least one element of the list achieves score s and length $t - \Delta$ where Δ is a positive integral parameter. We show that the dynamic programming formulation can be extended to preserve this property through the nodes. In particular, an alignment with score $\geq LAt^*$ and length $\geq t - \Delta$ will be observed in one of the nodes (i, j) during the computations. We imagine the vertices of $G_{X,Y}$ as grouped into $\lfloor (n + m)/\Delta \rfloor$ diagonal slabs at distance Δ from each other as shown in Figure 76.4.

Since we define the length of an alignment as the sum of the lengths of the substrings involved in the alignment, on a given alignment the contribution of each diagonal arc to the alignment length is 2 (each match, or mismatch involves two symbols, one from each sequence), while that of each horizontal or vertical arc is 1 (each indel involves one symbol from one of the sequences). Equivalently we say that the length of a diagonal arc is 2, and the length of each horizontal, or vertical arc is 1. The length of an alignment a is the total length of the arcs on a. Each slab consists of $\lfloor \Delta/2 \rfloor + 1$ diagonals. Two consecutive slabs share a diagonal which we call a *boundary*. The *left* and the *right boundaries* of slab b are, respectively, the boundaries shared by the left and right neighboring slabs of b. As a subgraph, a slab contains all the

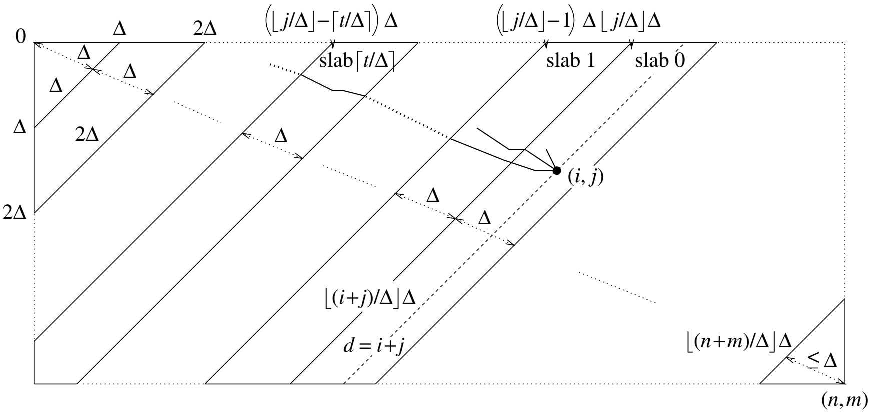


FIGURE 76.4 Slabs with respect to diagonal d and alignments ending at node (i, j) starting at different slabs.

edges in $G_{X,Y}$ incident to the vertices in the slab except for the horizontal and vertical edges incident to the vertices on the left boundary (which belong to the preceding slab), and the diagonal edges incident to the vertices on the first diagonal following the left boundary.

Now to a given diagonal d in $G_{X,Y}$, we associate a number of slabs as follows. Let *slab 0 with respect to diagonal d* be the slab that contains the diagonal d itself. The slabs to the left of slab 0 are then ordered, consecutively, as slab 1, slab 2, ... with respect to d. In other words, slab k with respect to diagonal d is the subgraph of $G_{X,Y}$ composed of vertices placed inclusively between diagonals $\lfloor d/\Delta \rfloor$ and d if $k = 0$, and between diagonal $(\lfloor d/\Delta \rfloor - k)\Delta$ and $(\lfloor d/\Delta \rfloor - k + 1)\Delta$, otherwise. Figure 76.4 includes sample slabs with respect to diagonal d, and alignments ending at some node (i, j) on this diagonal.

Let $\mathcal{S}_{i,j,k}$ represent the optimum score achievable at (i, j) by any alignment starting at slab k with respect to diagonal $i + j$ for $0 \le k < \lceil t/\Delta \rceil$. For $k = \lceil t/\Delta \rceil$, $\mathcal{S}_{i,j,k}$ is slightly different: It is the maximum of all achievable scores by an alignment starting in or before slab k. Also let $\mathcal{L}_{i,j,k}$ be the length of an optimal alignment starting at slab k, and achieving score $\mathcal{S}_{i,j,k}$. A single slab can contribute at most Δ to the length of any alignment. We store at each node (i, j), $\lceil t/\Delta \rceil + 1$ score–length pairs $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k})$ for $0 \le k \le \lceil t/\Delta \rceil$ corresponding to $\lceil t/\Delta \rceil + 1$ optimal alignments that end at (i, j). Figure 76.5 shows the steps of the algorithm *APX-LAt*. The processing is done row-by-row starting with the top row $(i = 0)$ of $G_{X,Y}$.

Step 1 of the algorithm performs the initialization of the lists of the nodes in the top row $(i = 0)$. Step 2 implements computation of scores as dictated by the dynamic programming formulation in Eq. (76.2). Let maxp of a list of score–length pairs be a pair with the maximum score in the list. We obtain an optimal alignment with score $\mathcal{S}_{i,j,k}$ by extending an optimal alignment from one of the nodes $(i - 1, j)$, $(i - 1, j - 1)$, or $(i, j - 1)$. We note that extending an alignment at (i, j) from node $(i - 1, j - 1)$ increases

```
Algorithm $APX\text{-}LAt(\delta, \mu)$
1. Initialization:  set $\widehat{LAt} = 0$; and $(\mathcal{S}_{0,j,k}, \mathcal{L}_{0,j,k}) = (0, 0)$ for all $j, k$,  $0 \le j \le m$, and $0 \le k \le \lceil t/\Delta \rceil$
2. Main computations:
   for $i = 1$ to $n$ do {
      set $(\mathcal{S}_{i,0,k}, \mathcal{L}_{i,0,k}) = (0, 0)$ for all $k$, $0 \le k \le \lceil t/\Delta \rceil$
      for $j = 1$ to $m$ do {
         if $(i + j$ mod $\Delta = 1)$ then {
            set $(\mathcal{S}_{i,j,0}, \mathcal{L}_{i,j,0}) = (0, 0)$
            for $k = 1$ to $\lceil t/\Delta \rceil - 1$ do
2.a.1          set $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k}) = \mathrm{maxp}\{\ (0, 0),\ (\mathcal{S}_{i-1,j,k-1}, \mathcal{L}_{i-1,j,k-1}) + (-\mu, 1),$
                        $(\mathcal{S}_{i-1,j-1,k-1}, \mathcal{L}_{i-1,j-1,k-1}) \oplus (s(x_i, y_j), 2),$
                        $(\mathcal{S}_{i,j-1,k-1}, \mathcal{L}_{i,j-1,k-1}) + (-\mu, 1)\ \}$
            for $k = \lceil t/\Delta \rceil$
2.a.2          set $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k}) = \mathrm{maxp}\{\ (0, 0),\ (\mathcal{S}_{i-1,j,k-1}, \mathcal{L}_{i-1,j,k-1}) + (-\mu, 1),$
                        $(\mathcal{S}_{i-1,j-1,k-1}, \mathcal{L}_{i-1,j-1,k-1}) \oplus (s(x_i, y_j), 2),$
                        $(\mathcal{S}_{i,j-1,k-1}, \mathcal{L}_{i,j-1,k-1}) + (-\mu, 1), (\mathcal{S}_{i-1,j,k}, \mathcal{L}_{i-1,j,k}) + (-\mu, 1),$
                        $(\mathcal{S}_{i-1,j-1,k}, \mathcal{L}_{i-1,j-1,k}) \oplus (s(x_i, y_j), 2), (\mathcal{S}_{i,j-1,k}, \mathcal{L}_{i,j-1,k}) + (-\mu, 1)\ \}$
         } else {
               for $k = 0$ to $\lceil t/\Delta \rceil$ do
2.b               set $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k}) = \mathrm{maxp}\{\ (0, 0),\ (\mathcal{S}_{i-1,j,k}, \mathcal{L}_{i-1,j,k}) + (-\mu, 1),$
                        $(\mathcal{S}_{i-1,j-1,k}, \mathcal{L}_{i-1,j-1,k}) \oplus (s(x_i, y_j), 2), (\mathcal{S}_{i,j-1,k}, \mathcal{L}_{i,j-1,k}) + (-\mu, 1)\ \}$
            }
            for $k = \lceil t/\Delta \rceil - 1$ if $\mathcal{L}_{i,j,k} \ge t - \Delta$ then set $\widehat{LAt} = \max\{\widehat{LAt}, \mathcal{S}_{i,j,k}\}$
            for $k = \lceil t/\Delta \rceil$ set $\widehat{LAt} = \max\{\widehat{LAt}, \mathcal{S}_{i,j,k}\}$
   } }
3. Return $\widehat{LAt}$
```

FIGURE 76.5 Algorithm *APX-LAt* [4].

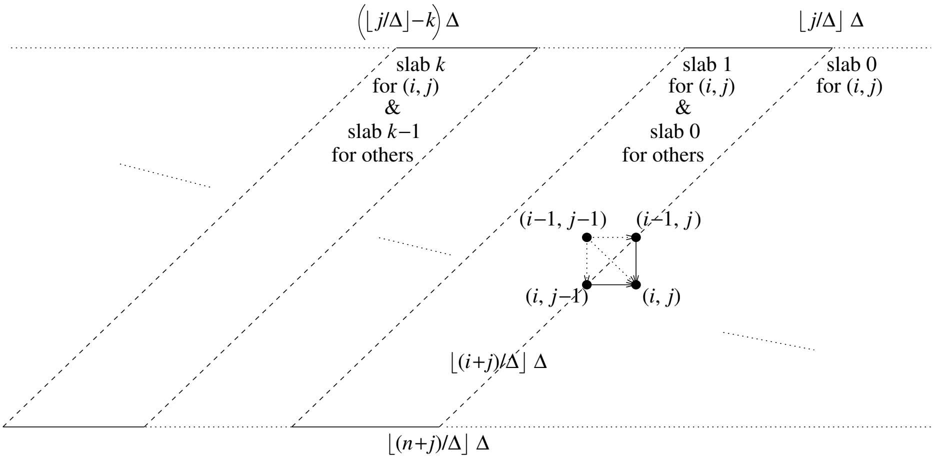


FIGURE 76.6 Relative numbering of the slabs with respect to $(i,\ j)$, $(i-1,\ j)$, $(i-1,\ j-1)$, and $(i,\ j-1)$ when node $(i,\ j)$ is on the first diagonal following boundary $\lfloor (i+j)/\Delta \rfloor$.

the length by 2 and the score by $s(x_i,\ y_j)$, whereas from nodes $(i-1,\ j)$ or $(i,\ j-1)$ adds 1 to the length and $-\mu$ to the score of the resulting alignment. There are two cases:

Case 1. If the current node $(i,\ j)$ is not on the first diagonal after a boundary then nodes $(i-1,\ j)$, $(i-1,\ j-1)$, and $(i,\ j-1)$ share the same slabs with node $(i,\ j)$. In this case $(\mathcal{S}_{i,\,j,k},\ \mathcal{L}_{i,\,j,k})$ is calculated by using $(\mathcal{S}_{i-1,\,j,k},\ \mathcal{L}_{i-1,\,j,k})$, $(\mathcal{S}_{i-1,\,j-1,k},\ \mathcal{L}_{i-1,\,j-1,k})$, and $(\mathcal{S}_{i,\,j-1,k},\ \mathcal{L}_{i,\,j-1,k})$ as shown in Step 2.*b*, where $(\mathcal{S}_{i-1,\,j-1,k},\ \mathcal{L}_{i-1,\,j-1,k}) \oplus (s(x_i,\ y_j),\ 2) = (\mathcal{S}_{i-1,\,j-1,k} + s(x_i,\ y_j),\ \mathcal{L}_{i-1,\,j-1,k} + 2)$ if $\mathcal{S}_{i-1,\,j-1,k} > 0$ or $k = 0$; and (0, 0) otherwise. This is because, by definition, a local alignment must have a positive score to exist, and it is either a single match, or it is an extension of an alignment whose score is positive. Therefore we do not let an alignment with zero score be extended. A new alignment starts with a single match in the current slap.

Case 2. If the current node is on the first diagonal following a boundary (i.e., $i + j \bmod \Delta = 1$) then the slabs for the nodes involved in the computations for node $(i,\ j)$ differ as shown in Figure 76.6. In this case slab k for node $(i,\ j)$ is slab $k-1$ for nodes $(i-1,\ j)$, $(i-1,\ j-1)$, and $(i,\ j-1)$. Moreover any alignment ending at $(i,\ j)$ starting at slab 0 for $(i,\ j)$ can only include one of the edges $((i-1,\ j),\ (i,\ j))$ or $((i-1,\ j-1),\ (i,\ j))$ both of which have negative weight $-\mu$. Therefore, $(\mathcal{S}_{i,\,j,0},\ \mathcal{L}_{i,\,j,0})$ is set to (0, 0). Steps 2.*a*.1 and 2.*a*.2 show the calculation of $(\mathcal{S}_{i,\,j,k},\ \mathcal{L}_{i,\,j,k})$ respectively for $0 < k < \lceil t/\Delta \rceil$ and for $k = \lceil t/\Delta \rceil$.

The running maximum score $\widehat{LAt}$ is updated whenever a newly computed score for an alignment with length $\geq t - \Delta$ is larger than the current maximum which can only happen with alignments starting in or before slab $\lceil t/\Delta \rceil - 1$. The final value $\widehat{LAt}$ is returned in Step 3. The alignment position achieving this score may also be desired. This can be done by maintaining for each optimal alignment a start and end position information besides its score and length. In this case in addition to the running maximum score, the start and end positions of a maximal alignment should be stored and updated.

We first show that $\mathcal{S}_{i,\,j,k}$ calculated by the algorithm is the optimum score achievable and $\mathcal{L}_{i,\,j,k}$ is the length of an alignment achieving this score over the set of all alignments ending at node $(i,\ j)$ and starting with respect to diagonal $i + j$: (1) at slab k for $0 \leq k < \lceil t/\Delta \rceil$ and (2) in or before slab k for $k = \lceil t/\Delta \rceil$. This claim can be proved by induction. If we assume that the claim is true for nodes $(i-1,\ j)$, $(i-1,\ j-1)$, and $(i,\ j-1)$, and for their slabs, then we can easily see by following Step 2 of the algorithm that the claim holds for node $(i,\ j)$ and its slabs.

Let optimum score LAt^* for the alignments of length $\geq t$ be achieved at node $(i,\ j)$. Consider the calculations of the algorithm at $(i,\ j)$ at which an optimal alignment ends. There are two possible orientations

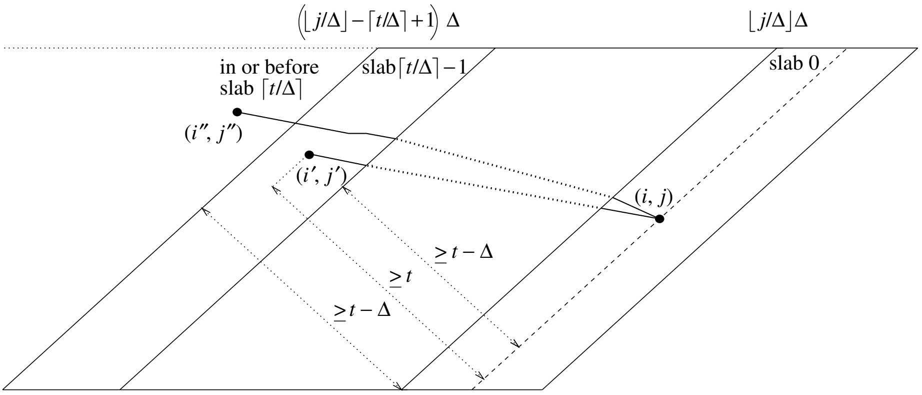

$(\lfloor j/\Delta \rfloor - \lceil t/\Delta \rceil + 1)\,\Delta$
$\lfloor j/\Delta \rfloor \Delta$
in or before slab $\lceil t/\Delta \rceil$
slab $\lceil t/\Delta \rceil - 1$
slab 0
(i'', j'')
(i', j')
(i, j)
$\geq t - \Delta$
$\geq t$
$\geq t - \Delta$

FIGURE 76.7 Two possible orientations of an optimal alignment of length $\geq t$ ending at (i, j): It starts either at some (i', j') at slab $\lceil t/\Delta \rceil - 1$, or (i'', j'') in or before slab $\lceil t/\Delta \rceil$.

of an optimal alignment as shown in Figure 76.7: (1) It starts at some node (i', j') of slab $k = \lceil t/\Delta \rceil - 1$. By a previous claim an alignment starting at slab k with score $\mathcal{S}_{i,j,k} \geq LAt^*$ is captured in Step 2. The length of this alignment $\mathcal{L}_{i,j,k}$ is at least $t - \Delta$ since the length of the optimal alignment is $\geq t$, and both start at the same slab and end at (i, j). (2) It starts at some node (i'', j'') in or before slab $k = \lceil t/\Delta \rceil$. Again by the previous claim an alignment starting in or before slab k with score $\mathcal{S}_{i,j,k} \geq LAt^*$ is captured in Step 2. The length of this alignment $\mathcal{L}_{i,j,k}$ is at least $t - \Delta$ since slab k is at distance $\geq t - \Delta$ from (i, j). Therefore the final value $\widehat{LAt}$ returned in Step 3 is $\geq LAt^*$ and it is achieved by an alignment whose length is $\geq t - \Delta$. We summarize these results in the following theorem:

Theorem 76.1 (Arslan and Eğecioğlu [4])

For a feasible LAt problem, Algorithm APX-LAt returns an alignment $(\widehat{I}, \widehat{J})$ such that $s(\widehat{I}, \widehat{J}) \geq LAt^*$ and $|\widehat{I}| + |\widehat{J}| \geq (1 - \frac{1}{r})t$ for any r, $1 < r \leq t/2$. The algorithm's complexity is $O(rnm)$ time and $O(rm)$ space.

Proof

Algorithm *APX-LAt* is similar to the Smith–Waterman algorithm except that at each node instead of a single score, $\lceil t/\Delta \rceil + 1$ entries for score–length pairs are stored and manipulated. Therefore the resulting complexity exceeds that of the Smith–Waterman algorithm by a factor of $\lceil t/\Delta \rceil + 1$. That is, the time complexity of *APX-LAt* is $O(nmt/\Delta)$. The algorithm requires $O(mt/\Delta)$ space since the computations proceed row by row, and we need the entries in the previous and the current row to calculate the entries in the current row. When the *LAt* problem is feasible, it is guaranteed that Algorithm *APX-LAt* returns an alignment $(\widehat{I}, \widehat{J})$ such that $s(\widehat{I}, \widehat{J}) \geq LAt^* > 0$ and $|\widehat{I}| + |\widehat{J}| \geq t - \Delta$ for any positive Δ. Therefore setting $\Delta = \lfloor t/r \rfloor$ for a choice of r, $1 < r \leq t/2$, and using Algorithm *APX-LAt* we can achieve the approximation and complexity results expressed in the theorem. We also note that for $\Delta = 2$ the algorithm becomes a dynamic programming algorithm extending the dimension by storing all possible alignment lengths. □

A variant of *APX-LAt* for arbitrary scoring matrices can be obtained by simple modifications: At each entry of the dynamic programming matrix, instead of a single score a number of scores (and lengths) are maintained and manipulated as dictated by the underlying dynamic programming formulation (e.g., Eq. [76.4]).

An application of the *LAt* problem is on problem *Qt* which is defined as the problem of finding two subsequences with normalized score higher than λ, and total length at least *t*. More formally

$$Qt:\ \text{find } (I, J) \text{ such that } I \subseteq X,\ J \subseteq Y,\ \frac{s(I, J)}{|I| + |J|} > \lambda \quad \text{and} \quad |I| + |J| \geq t \tag{76.10}$$

The following simple query explains the motivation for this problem: "Do two sequences share a (sufficiently long) fragment with more than 70% of similarity?"

The problem is feasible for given thresholds t, and $\lambda > 0$, if the answer to this query is not empty, that is, there exists a pair of subsequences I and J with total length $|I| + |J| \geq t$, and normalized score $s(I, J)/(|I| + |J|) > \lambda$. Note that Qt is feasible iff $NLAt^* > \lambda$. We describe an algorithm which returns for a feasible problem two subsequences $\widehat{I} \subseteq X$ and $\widehat{J} \subseteq Y$ with normalized score higher than λ, and total length $|\widehat{I}| + |\widehat{J}| \geq (1 - \frac{1}{r})t$. The approximation ratio is controlled by parameter r. The computations take $O(rnm)$ time and $O(rm)$ space.

For a given λ, we define *the parametric local alignment with length threshold problem LAt*(λ) as follows:

$$LAt(\lambda) : \text{maximize } SCORE(a) - \lambda\ LENGTH(a) \quad \text{s.t. } a \in AVt$$

Proposition 76.2 **(Arslan and Eğecioğlu [4])**

For $\lambda < \frac{1}{2}$, the optimum value $LAt^*(\lambda)$ of the parametric LAt problem can be formulated in terms of the optimum value LAt^* of an LAt problem.

Proof

The proof is very similar to that of Proposition 76.1. Under the simple scoring scheme the optimum value of the parametric problem, when $\lambda < \frac{1}{2}$, is

$$LAt^*_{\delta,\mu}(\lambda) = (1 - 2\lambda)LAt^*_{\delta',\mu'}, \text{ where } \delta' = \frac{\delta + 2\lambda}{1 - 2\lambda},\ \mu' = \frac{\mu + \lambda}{1 - 2\lambda} \qquad (76.11)$$

We can easily see that a similar relation exists in the case of arbitrary scoring matrices, and affine gap penalties. Computing $LAt^*(\lambda)$ involves solving the local alignment with length threshold problem LAt and performing some simple arithmetic afterward. □

Under the scoring schemes we study we assume without loss of generality that for any alignment, its normalized score is $\leq \frac{1}{2}$. We consider $\lambda = \frac{1}{2}$ as a special case which can only happen when the alignment is composed of matches only.

Proposition 76.3 **(Arslan and Eğecioğlu [4])**

When solving LAt(λ), the underlying algorithm for LAt returns an alignment $(\widehat{I}, \widehat{J})$ with normalized score higher than λ, and $|\widehat{I}| + |\widehat{J}| \geq (1 - \frac{1}{r})t$ if problem Qt is feasible.

Proof

Assume that problem Qt is feasible. Then $LAt^*(\lambda) > 0$, which implies that the algorithm which solves the corresponding LAt problem (of Proposition 76.2) returns an alignment $(\widehat{I}, \widehat{J})$ such that its score is positive (i.e., $s(\widehat{I}, \widehat{J}) - \lambda(|\widehat{I}| + |\widehat{J}|) > 0$) and $|\widehat{I}| + |\widehat{J}| \geq (1 - \frac{1}{r})t$ by the approximation results of Algorithm APX-LAt. □

Thus solving Qt requires a single application of Algorithm APX-LAt.

76.5 Normalized Local Alignment

Need for a length constraint is clear when length-normalized scores are used because shorter alignments may have high normalized scores but they may not be biologically significant. The definition of the $NLAt$ problem contains a length constraint as described in Section 76.3.

Let $AVt \subseteq AV$ be the set of alignment vectors corresponding to alignments with length $\geq t$. The normalized local alignment problem $NLAt$ can be rewritten as follows:

$$NLAt : \text{maximize } \frac{SCORE(a)}{LENGTH(a)} \quad \text{s.t. } a \in AVt$$

```
Algorithm $APX\text{-}RationalNLAt$
If there is an exact match of size $(1-\frac{1}{r})t$ then return($\frac{1}{2}$) and exit
Let $\sigma$ be the smallest gap between two length-normalized scores
$[e,\, f] \leftarrow [0,\, \frac{1}{2}\sigma^{-1}]$
$\lambda^* \leftarrow 0$
While ($e + 1 < f$) do
  $k \leftarrow \lceil (e + f)/2 \rceil$
  $APX\text{-}LAt^*(k\sigma) > 0$ then {
    $e \leftarrow k$
    $\lambda^* \leftarrow$ the normalized score of an optimal alignment obtained
  } else $f \leftarrow k$
End {while}
Return($\lambda^*$)
```

FIGURE 76.8 Algorithm *APX-RationalNLAt* for rational scores [4].

We present next approximation algorithms for the *NLAt* problem that apply fractional programming in which we use Algorithm *APX-LAt* as a subroutine. The approximation is in the sense that the length constraint is partially satisfied. These algorithms are the *Dinkelbach* algorithm for *NLAt*, and Algorithm *RationalNLAt*. Both algorithms obtain an alignment whose score is no smaller than the optimum score *NLAt** of the original *NLAt* problem, and whose length is at least $(1 - \frac{1}{r})t$ for a given r provided that the original *NLAt* problem is feasible (Theorem 76.2). The Dinkelbach algorithm for *NLAt* (Figure 76.9) and Rational*NLAt* (Figure 76.8) are similar to the corresponding *ANLA* algorithms except that they iteratively solve *LAt* problems presented in Section 76.4 instead of *LA* problems. The approximation algorithm *APX-LAt* can be applied to solving the parametric problems that arise in computing *NLAt**.

In both resulting algorithms the space complexity is $O(rm)$. The observed time complexity of the *Dinkebach* algorithm for *NLAt* is $O(rnm)$ (in tests [4], it performs always smaller than 10, and on average 3–5 invocations to Algorithm *APX-LAt*). Algorithm *RationalNLAt* has proven time complexity $O(rnm \log\ n)$ since in this algorithm $O(\log\ n)$ invocations of *APX-LAt* is sufficient to solve the *NLAt* problem.

We reiterate the definitions of the local alignment with length threshold *LAt*, normalized local alignment *NLAt*, and the parametric local alignment $LAt(\lambda)$ problems as the following optimization problems defined in terms of *SCORE* and *LENGTH* functions that are linear over *AVt* under the scoring schemes we study:

$$\begin{array}{lll} LAt: & \text{maximize } SCORE(a) & \text{s.t. } a \in AVt \\ NLAt: & \text{maximize } \dfrac{SCORE(a)}{LENGTH(a)} & \text{s.t. } a \in AVt \\ LAt(\lambda): & \text{maximize } SCORE(a) - \lambda\ LENGTH(a) & \text{s.t. } a \in AVt \end{array}$$

If we apply the fractional programming to the normalized local alignment computation then we can obtain an optimal solution to *NLAt* via a series of optimal solutions of the parametric problem with different parameters $LAt(\lambda)$ such that $\lambda = NLAt^*$ iff $LAt^*(\lambda) = 0$.

Theorem 76.2 (Arslan and Eğecioğlu [4])

If $NLAt^* > 0$ *then an alignment with normalized score at least* $NLAt^*$*, and total length at least* $(1 - \frac{1}{r})t$ *can be computed for any* r*,* $1 < r \leq t/2$ *in time* $O(rnm \log n)$ *and space* $O(rm)$*.*

Proof

Algorithm *Rational NLAt* given in Figure 76.8 accomplishes this. The algorithm is based on a binary search for optimum-normalized score over an interval of integers. This takes $O(\log n)$ parametric problems to solve. The algorithm is similar to the *RationalANLA* algorithm in Figure 76.3, and the results are derived similarly. It first determines if there is an exact match of size $(1 - \frac{1}{r})t$, which can easily be done by using the Smith–Waterman algorithm. If the answer is yes then the algorithm returns the maximum possible normalized score and exits. The skeleton of the rest of the algorithm is the same as Algorithm *RationalNLAt* in Figure 76.3, based on Megiddo's search technique [9]. The difference is that the parametric alignment

```
Algorithm Dinkelbach
If $APX\text{-}LAt^*(0) > 0$ then set $\lambda^*$ to the length-normalized score of an optimal alignment else exit
Repeat
    $\lambda \leftarrow \lambda^*$
    If $APX\text{-}LAt^*(\lambda) > 0$ then set $\lambda^*$ to the length-normalized score of an optimal alignment
Until $\lambda^* \leq \lambda$
Return($\lambda^*$)
```

FIGURE 76.9 Dinkelbach algorithm for *NLAt* [4].

problems now have a length constraint. The algorithm computes the smallest possible gap σ between any two distinct possible normalized scores, which is $\Omega(1/(n+m)^2)$ [2]. It maintains an interval $[e, f]$, on which a binary search is done to find the largest λ for which $LAt^*(\lambda)$ is positive where e and f are integer variables. Initially e is set to 0, and f is set to $\frac{1}{2}\sigma^{-1}$ since $NLAt^*$ is in $[0, \frac{1}{2}]$. A parametric *LAt* problem with parameter $k\sigma$ is iteratively solved, where k is the median of integers in $[e, f]$. At each iteration the interval is updated according to the sign of the value of the parametric problem. The effective search space is the integers in $[e, f]$ and each iteration reduces this space by half. The iterations end whenever there remains no integer between e and f. By Theorem 76.1 and Proposition 76.3 in Section 76.4 for every $k\sigma < NLAt^*$, Algorithm *APX-LAt* returns an alignment with a positive score, and length at least $(1 - \frac{1}{r})t$ as a solution to the parametric problem. After the search ends, $\lambda^* \geq NLAt^*$, and λ^* is achieved by an alignment whose length is at least $(1 - \frac{1}{r})t$ for *NLAt* feasible. Note that if $NLAt^* = 0$ then the algorithm returns 0.

The asymptotic space requirement is the same as that of Algorithm *APX-LAt*, and the loop iterates $O(\log n)$ times. Therefore the complexity results are as described in the theorem. □

If $NLAt^* > 0$ then we can also achieve the same approximation guarantee by using the Dinkelbach algorithm given by Arslan et al. [2] as the template. The details of the resulting algorithm appear in Figure 76.9. At each iteration, except for the last, Algorithm *APX-LAt* returns an alignment with a positive score, and length at least $(1 - \frac{1}{r})t$ as a solution to the parametric problem by Theorem 76.1 and Proposition 76.3 in Section 76.4 since $\lambda < NLAt^*$. Solutions of the parametric problems through the iterations yield improved (higher) values to λ except for the last iteration. The resulting algorithm performs no more than 3–5 iterations on average, and never more than 9 in the worst case in tests [4]. When the algorithm terminates, the optimal alignment whose length-normalized score is λ^* has the total length at least $(1 - \frac{1}{r})t$ and $\lambda^* \geq NLAt^*$.

76.6 Discussion

We would like to point out the relation between the normalized local alignment and a problem known as *parametric sequence alignment* [10] (which is different from the parametric local alignment problem we discuss in this chapter) in the literature. The fractional programming-based *ANLA* and *NLAt* algorithms iteratively, and systematically change the four parameters (i.e., match score, mismatch, gap open, and gap extension penalties) until the resulting alignment is satisfactory (i.e., optimal both with respect to ordinary scores at the last iteration and with respect to length-normalized scores with the original scores). It has been known that sequence alignment is sensitive to the choice of these parameters as they change the optimality of the alignments. Parametric sequence alignment studies the relation between the parameter settings and optimal alignments. The goal is to partition the parameter space into convex polygons such that the same alignment is optimal at every point in the same polygon. Clearly a point in one of the polygons computed yields an optimal length-normalized alignment. The following results are summarized by Gusfield [11]: A polygonal decomposition requires $O(nm)$ time per polygon when scores are uniform (i.e., not dependent on individual symbols). When only two parameters are chosen to be variable then the polygonal decomposition can contain at most $O(nm)$ polygons. When all the four parameters are variables then there is no known reasonable upper bound on the number of polygons. When the alignment is global, and no character-specific scoring matrices are used the number of polygons is bounded from above by $O(n^{2/3})$ [12].

We also remark that to find long regions with high degree of similarity we may also formulate an objective with which we aim to minimize a length-normalized weighted edit distance for substrings, and include a length threshold as a lower bound for the desired length. For solving this problem Karp's $O(|V||E|)$-time minimum mean-weight cycle algorithm [13] seems a natural candidate. This solution requires adding extra edges to cause cycles of minimum certain length determined by the given length threshold. For an alignment graph for a pair of strings of length n each, the number of vertices $|V|$ and number of edges $|E|$ (excluding the additional edges) are both $O(n^2)$. This is not more efficient than the naive dynamic programming solution.

We conclude by stating a few open problems for further study:

> *How many iterations do the Dinkelbach ANLA or NLAt algorithms perform in the worst case?*
>
> *Are there (provably) faster exact or better approximation algorithms for the NLA, LRLA, LAt, or Qt problems?*

References

[1] Arslan, A. N. and Eğecioğlu, Ö., An improved upper bound on the size of planar convex-hulls, *Proc. of COCOON,* Lecture Notes in Computer Science, 2108, 2001, p. 111.

[2] Arslan, A. N., Eğecioğlu, Ö., and Pevzner, P. A., A new approach to sequence comparison: normalized local alignment, *Bioinformatics*, 17(4), 327, 2001.

[3] Arslan, A. N. and Eğecioğlu, Ö., Approximation algorithms for local alignment with length constraints, *Int. J. Found. Comput. Sci.*, 13(5), 751, 2002.

[4] Arslan, A. N. and Eğecioğlu, Ö., Dynamic programming based approximation algorithms for local alignment with length constraints, *INFORMS J. Comput., Special Issue on Comput. Mol. Biol./Bioinf.*, 16(4), 441, 2004.

[5] Waterman, M. S., *Introduction to Computational Biology*, Chapman & Hall/CRC, New York, 1995.

[6] Smith, T. F. and Waterman, M. S., The identification of common molecular subsequences, *J. Mol. Biol.*, 147(1), 195, 1981.

[7] Craven, B. D., *Fractional Programming*, Helderman Verlag, Berlin, 1988.

[8] Sniedovich, M., *Dynamic Programming*, Marcel Dekker, New York, 1992.

[9] Megiddo, N., Combinatorial optimization with rational objective functions, *Math. Oper. Res.*, 4, 414, 1979.

[10] Fitch, W. M. and Smith, T. F., Optimal sequence alignments, in *Proc. Natl. Acad. Sci.*, 80, 1983, 1382.

[11] Gusfield, D., *Algorithm on Strings, Trees, and Sequences: Computer Science and Computational Biology*, The Press Syndicate of The University of Cambridge, New York, 1997.

[12] Gusfield, D., Balasubramanian, K. and Naor, D., Parametric optimization of sequence alignment, *Algorithmica*, 12, 312, 1994.

[13] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C., *Introduction to Algorithms*, 2nd ed., The MIT Press, Cambridge, MA, 2001.

77

Approximation Algorithms for the Selection of Robust Tag SNPs

Yao-Ting Huang
National Taiwan University

Kui Zhang
University of Alabama at Birmingham

Ting Chen
University of Southern California

Kun-Mao Chao
National Taiwan University

77.1 Introduction

Each individual has a unique set of genetic blueprints stored in a long and spiral-shaped molecule called *deoxyribonucleic acid* (DNA). The genetic blueprints are composed of linked subunits called *nucleotides*. Each nucleotide carries one of the four genetic codes: adenine (A), cytosine (C), guanine (G), and thymine (T). The variations of genetic codes from individual to individual (e.g., insertions, deletions, and mutations) have a major impact on genetic diseases and phenotypic differences. Therefore, correlating genetic variations with diseases or traits is the next important step in human genomics. In the following, we first introduce the related biological background to understand the problem studied in this chapter. Then we describe a frequently encountered problem in the current experimental environment, which is the main focus of this chapter.

77.1.1 Single Nucleotide Polymorphisms and Haplotypes

Among various genetic variations, *single nucleotide polymorphisms* (SNPs) are generally considered to be the most frequent form, which has fundamental importance for genetic disease association and drug design. A SNP is a genetic variation when a single nucleotide (i.e., A, C, G, or T) in the genomic sequence is altered and kept through heredity thereafter.[1] It has been shown that approximately 90% of genetic variations are made up of SNPs. Up to the present, millions of SNPs have been identified and these data are now publicly available for researchers [1,2]. The SNPs can be further divided into the following types depending on their region and function to the amino acid sequence.

[1] The genetic variation is considered to be a SNP only if it is observed with frequency at least 1% in the population. Otherwise, it is considered to be a mutation.

Coding SNP (cSNP). A SNP in the coding region that involves in the regulation of amino acid substitution.

Synonymous SNP (sSNP). A cSNP that synonymously changes the codon of an amino acid, which does not alter the amino acid sequence.

Nonsynonymous SNP (nsSNP). A cSNP that nonsynonymously changes the codon of an amino acid, which alters the amino acid sequence.

Despite many types of SNPs, almost all SNPs observed have only two variants called *alleles*. Very few SNPs (about 0.1%) in the human population have been found to have more than two different nucleotides. In fact, these third type nucleotides are often resulted from possible experimental errors [3]. Consequently, most studies usually assume that the value of a SNP is binary. A SNP is referred to as the *major allele* if it is the wild type and is called the *minor allele* otherwise (i.e., mutant type). In this chapter, each type of SNP is equally treated as a binary variable.

A set of linked SNPs on one genomic sequence is referred to as a *haplotype*. Because the value of a SNP is usually assumed to be binary, a haplotype can be simply considered to be a binary string. *Linkage disequilibrium* (LD), which refers to the nonrandom association of alleles at different loci in haplotypes, plays an important role in genome-wide association studies for identifying genetic variations responsible for common diseases. A number of studies have shown that using haplotypes instead of individual SNP as the basic units for LD analysis can greatly reduce the noise [4]. Recently, the International HapMap Project [1], formed in 2002, aimed to characterize the patterns of LD across the human genome such that the information can be used for large-scale genetic association studies.

77.1.2 Haplotype Blocks and Tag SNPs

The LD analysis of haplotypes has greatly affected the frequency of past *recombination* events. Recombination (or cross over) is a process during meiosis that the two homologous chromosomes (inside the cells that produce sperms or eggs) break and swap portion with each other. After these two homologous chromosomes glue themselves back, each of them obtains new alleles from the other. Therefore, the result of recombination can produce new chromosomes for the offspring. However, the nonrandom association of alleles at different loci is broken up by the recombination occurred in between. Consequently, recombination can also reduce the LD observed in the population.

In recent years, the patterns of LD observed in the human population show a block-like structure [4–7]. The entire chromosome can be partitioned into high LD regions interspersed by low LD regions. The chromosome recombination almost only takes place at those low LD regions called recombination hotspots. The high LD region between recombination hotspots is often referred to as a "haplotype block." Within a haplotype block, there is little or even no recombination occurred, and the SNPs in the block tend to be inherited together. Due to the low haplotype diversity within a block, the information carried by these SNPs is highly redundant. Thus, a small subset of SNPs, called "tag SNPs," is sufficient to distinguish each pair of patterns in the block [5,7,8–12]. Haplotype blocks with corresponding tag SNPs are quite useful and cost-effective for association studies as it does not require genotyping all SNPs.

Many studies have tried to find the minimum set of tag SNPs. These studies can be classified into the following categories.

LD-bins-based model. These methods try to identify minimum bins of SNPs such that all SNPs in the same bin are in high LD (e.g., $r^2 \geq 0.8$) with each other (e.g., Carlson et al. [9]). Most of them solve some variants of the minimum clique cover problem [13].

Blocks-based model. These methods assume that the block partition is available as input, and try to find a minimum set of SNPs, which is able to distinguish each pair of haplotypes in a block (e.g., Zhang et al. [11]). Most of them solve some variants of the minimum test set problem [13].

Block-free-based model. These methods define an informative measure or prediction accuracy to evaluate a set of tag SNPs. Most of them try to find a minimum set of SNPs, which maximizes their criterion (e.g., Bafna et al. [8]).

In this chapter, we study the tag SNP selection problem following the blocks-based model. In a large-scale study of human chromosome 21, Patil et al. [5] developed a greedy algorithm to partition the haplotypes into 4,135 blocks with 4,563 tag SNPs. Zhang et al. [7,11,12] used a dynamic programming approach to reduce the numbers of blocks and tag SNPs to 2,575 and 3,562, respectively. Bafna et al. [8] showed that the general version of this problem is NP-hard and gave efficient algorithms for special cases of this problem. In the following, we show that the previous studies do not consider the influence of missing data in the current SNP detection environment.

77.1.3 The Problem of Missing Data

In reality, a SNP may not be genotyped and is considered to be missing data (i.e., we fail to obtain the allele configuration of the SNP) if it does not pass the threshold of data quality [5,7,14]. The missing rates of SNPs can be up to 10% under the current genotyping experiment. In practice, there could be two kinds of missing data: completely and partially missing data. In this chapter, partially missing data are handled in analogy to completely missing data. These missing data may cause ambiguity when using the minimum set of tag SNPs to distinguish an unknown haplotype sample. As a consequence, the power of using tag SNPs for association study is reduced by missing data.

Figure 77.1 illustrates the influence of missing data when using the minimum set of tag SNPs to identify haplotype samples. In this figure, a haplotype block (see Figure 77.1 [a]) defined by 12 SNPs and 4 haplotype patterns is presented (from the haplotype database of human chromosome 21 by Patil et al. [5]). We follow the same assumption as Patil et al. [5] and Bafna et al. [8] that all SNPs are biallelic (i.e., taking on only two values). Suppose we select SNPs S_1 and S_{12} as tag SNPs. The haplotype sample h_1 is identified as haplotype pattern P_3 unambiguously (see Figure 77.1 [b]). Consider haplotype samples h_2 and h_3 with one tag SNP genotyped as missing data (see Figure 77.1 [c]). Sample h_2 can be identified as haplotype patterns P_2 or P_3, and h_3 can be identified as P_1 or P_3. As a result, these missing tag SNPs result in ambiguity when identifying haplotype samples.

Although we cannot avoid the occurrence of missing data, the remaining SNPs within the haplotype block may provide abundant information to resolve the ambiguity. For example, if we regenotype an additional SNP S_5 for h_2 (see Figure 77.1 [d]), h_2 is identified as haplotype pattern P_3 unambiguously. However, if SNP S_8 is regenotyped (see Figure 77.1 [e]), h_3 is also identified unambiguously. These additional SNPs are referred to as "auxiliary tag SNPs," which can be found from the remaining SNPs in the block and are able to resolve the ambiguity caused by missing data.

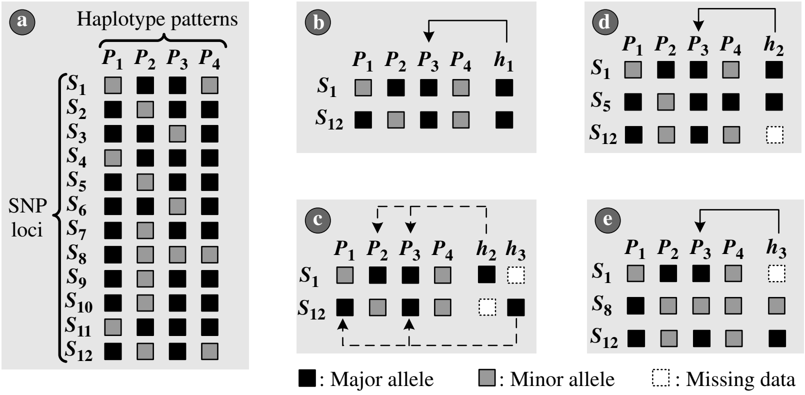

a
Haplotype patterns
P_1 P_2 P_3 P_4
SNP loci
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9 S_{10} S_{11} S_{12}
b
P_1 P_2 P_3 P_4 h_1
S_1 S_{12}
c
P_1 P_2 P_3 P_4 h_2 h_3
S_1 S_{12}
d
P_1 P_2 P_3 P_4 h_2
S_1 S_5 S_{12}
e
P_1 P_2 P_3 P_4 h_3
S_1 S_8 S_{12}
■: Major allele
▨: Minor allele
⬚: Missing data

FIGURE 77.1 The influence of missing data and auxiliary tag SNPs. (a) A haplotype block defined by 12 SNPs and 4 haplotype patterns. Each column represents a haplotype pattern and each row represents a SNP locus. The black and grey boxes stand for the major and minor alleles at each SNP locus, respectively. (b) Tag SNPs genotyped without missing data. (c) Tag SNPs genotyped with missing data. (d) The auxiliary tag SNP S_5 for h_2. (e) The auxiliary tag SNP S_8 for h_3.

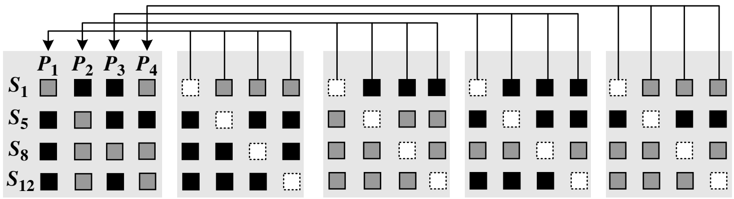


FIGURE 77.2 A set of robust tag SNPs for tolerating one missing tag SNP.

Alternatively, instead of regenotyping auxiliary tag SNPs whenever encountering missing data, we work on a set of SNPs, which is not affected by the the occurrence of missing data. Figure 77.2 illustrates a set of SNPs, which can tolerate one missing SNP. Suppose we select SNPs S_1, S_5, S_8, and S_{12} to be genotyped. Note that no matter which SNP is missing, each pair of patterns can still be distinguished by the remaining three SNPs. Therefore, all haplotype samples with one missing SNP can still be identified unambiguously. We refer to these SNPs as "robust tag SNPs," which are able to tolerate a certain number of missing data. The important feature of robust tag SNPs is that although they consume more SNPs than the "tag SNPs" defined in previous studies, they guarantee that all haplotype patterns with a certain number of missing data can be distinguished unambiguously. When the occurrence of missing data is frequent, the cost of regenotyping processes can be reduced by robust tag SNPs.

This chapter studies the problems of finding robust and auxiliary tag SNPs. Our study indicates that auxiliary tag SNPs can be found efficiently when robust tag SNPs have been computed in advance. This chapter is organized as follows. In Section 77.2, we show that the problem of finding minimum robust tag SNPs (MRTS) is NP-hard, and propose two greedy and one iterative linear programming (LP) relaxation algorithms, which find solutions of $(m+1)\ln(\frac{K(K-1)}{2})$, $\ln((m+1)\frac{K(K-1)}{2})$, and $O(m\ln K)$ approximation, respectively. Section 77.3 describes an efficient algorithm to find auxiliary tag SNPs when robust tag SNPs have been computed in advance. Section 77.4 presents the experimental results of our algorithms tested on a variety of simulated and biological data. Finally, concluding remarks are given in Section 77.5.

77.2 Finding Robust Tag SNPs

Assume we are given a haplotype block consisting of N SNPs and K haplotype patterns. This block is denoted by an $N \times K$ binary matrix M_h (see Figure 77.3 [a]). Define $M_h[i, j] \in \{1,2\}$ for each $i \in [1, N]$ and $j \in [1, K]$, where 1 and 2 represent the major and minor alleles, respectively.[2] Define C as the set of

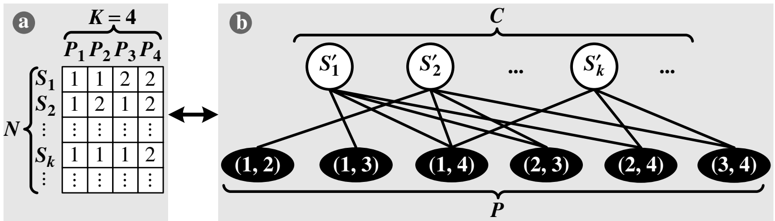


FIGURE 77.3 (a) The haplotype matrix M_h containing N SNPs and K haplotype patterns. (b) The bipartite graph corresponding to M_h.

[2] In reality, the haplotype block may also contain missing data. This formulation can be easily extended to handle missing data by treating them as "don't care" symbols. To simplify the presentation, we will assume no missing data in the block.

SNPs (i.e., rows) in M_h. The set of robust tag SNPs $C' \subseteq C$ (which allows up to m missing SNPs) must satisfy the following two properties: (1) an unknown haplotype sample can be identified (as one of the K patterns) by SNPs in C' unambiguously; (2) when at most m SNPs in C' are genotyped as missing data, (1) still holds. Note that to identify a sample unambiguously, each pair of patterns must be distinguished by at least one SNP in C'. For example (see Figure 77.3 [a]), patterns P_1 and P_2 can be distinguished by SNP S_2 since $M_h[2, 1] \neq M_h[2, 2]$. A formal definition of this problem is given below.

Problem: Minimum Robust Tag SNPs

Input. An $N \times K$ matrix M_h and an integer m.
Output. The minimum subset of SNPs $C' \subseteq C$ which satisfies:
(1) for each pair of patterns P_i and P_j, there is a SNP $S_k \in C'$ such that $M_h[k, i] \neq M_h[k, j]$;
(2) when at most m SNPs are discarded from C' arbitrarily, (1) still holds.

We then reformulate MRTS to a variant of the *set covering problem* [13]. Each SNP $S_k \in C$ (i.e., the k-th row in M_h) is reformulated to a set $S_k^{'} = \{(i, j) \mid M[k, i] \neq M[k, j] \text{ and } i < j\}$. For example, suppose the kth row in M_h is $\{1,1,1,2\}$. The corresponding set $S_k^{'} = \{(1, 4), (2, 4), (3, 4)\}$. In other words, $S_k^{'}$ stores pairs of patterns distinguished by SNP S_k. Define P as the set that contains all pairs of patterns (i.e., $P = \{(i, j) \mid 1 \leq i < j \leq K\} = \{(1, 2), (1, 3), \ldots, (K-1, K)\}$).

Consider each element in P and each reformulated set of C as nodes in an undirected bipartite graph (see Figure 77.3 [b]). If SNP S_k can distinguish patterns P_i and P_j (i.e., $(i, j) \in S_k^{'}$), there is an edge connecting the nodes (i, j) and $S_k^{'}$. The following lemma implies that each pair of patterns must be distinguished by at least $(m + 1)$ SNPs to allow m SNPs genotyped as missing data.

Lemma 77.1

$C' \subseteq C$ is the set of robust tag SNPs, which allows at most m SNPs genotyped as missing data iff each node in P has at least $(m + 1)$ edges connecting to each node in C'.

Proof

Let C' be the set of robust tag SNPs, which allows m SNPs genotyped as missing data. Suppose patterns P_i and P_j are distinguished by only m SNPs in C' (i.e., (i, j) has only m edges connecting to nodes in C'). However, if these m SNPs are genotyped as missing data, no SNPs in C' are able to distinguish patterns P_i and P_j, which is a contradiction. Thus, each pair of patterns must be distinguished by at least $(m + 1)$ SNPs, which implies that each node in P must have at least $(m + 1)$ edges connecting to nodes in C'. The proof of the other direction is similar. □

In the following, we give a lower bound on the minimum number of robust tag SNPs required.

Lemma 77.2

Given K haplotype patterns, the minimum number of robust tag SNPs for tolerating m missing SNPs is at least $m + \log K$.

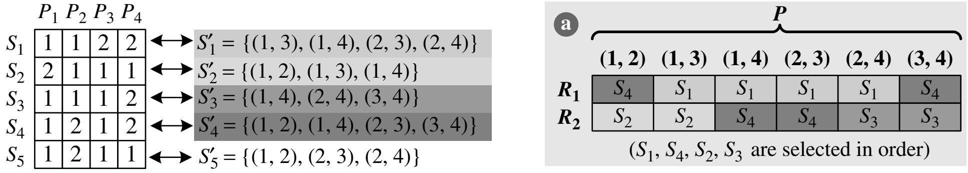
Proof

Recall that the value of a SNP is binary. The maximum number of distinct haplotypes, which can be distinguished by N SNPs is at most 2^N. As a result, to distinguish K distinct haplotype patterns, at least $\log K$ SNPs are required since $2^{\log K} = K$. In addition, there could be up to m missing SNPs. Therefore, the minimum number of robust tag SNPs required is at least $m + \log K$. □

Now we show the NP-hardness of the MRTS problem, which implies there is no polynomial-time algorithm to find the optimal solution of MRTS.

Theorem 77.1

The MRTS problem is NP-hard.



	P_1	P_2	P_3	P_4	
S_1	1	1	2	2	↔ $S'_1 = \{(1,3), (1,4), (2,3), (2,4)\}$
S_2	2	1	1	1	↔ $S'_2 = \{(1,2), (1,3), (1,4)\}$
S_3	1	1	1	2	↔ $S'_3 = \{(1,4), (2,4), (3,4)\}$
S_4	1	2	1	2	↔ $S'_4 = \{(1,2), (1,4), (2,3), (3,4)\}$
S_5	1	2	1	1	↔ $S'_5 = \{(1,2), (2,3), (2,4)\}$

	(1, 2)	(1, 3)	(1, 4)	(2, 3)	(2, 4)	(3, 4)
R_1	S_4	S_1	S_1	S_1	S_1	S_4
R_2	S_2	S_2	S_4	S_4	S_3	S_3

(S_1, S_4, S_2, S_3 are selected in order)

FIGURE 77.4 The SNPs S_1, S_4, S_2, and S_3 are selected by the first greedy algorithm. (a) The table that stores each selected SNP.

Proof

When $m = 0$, MRTS is the same as the original problem of finding minimum number of tag SNPs, which is known as the *minimum test set* problem [11,13]. Since the minimum test set problem is NP-hard and can be reduced to a special case of MRTS, MRTS is NP-hard. □

77.2.1 The First Greedy Algorithm

To solve MRTS efficiently, we propose a greedy algorithm, which returns a solution with a number of SNPs that is not extremely far from optimal. By Lemma 77.1, to tolerate m missing tag SNPs, we need to find a subset of SNPs $C' \subseteq C$ such that each pair of patterns in P is distinguished by at least $(m+1)$ SNPs in C'. Assume that the SNPs selected by this algorithm are stored in a $(m+1)\times|P|$ table (see Figure 77.4 [a]). Initially, each grid in the table is empty. Once a SNP S_k (that can distinguish patterns P_i and P_j) is selected, one grid of the column (i, j) is filled in with S_k, and we say that this grid is *covered* by S_k.

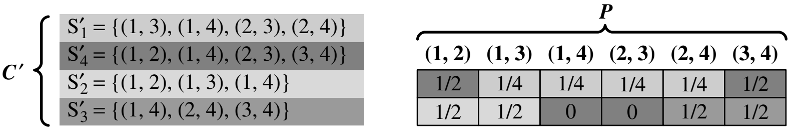
This greedy algorithm works by covering the grids from the first row to the $(m+1)$th row, and greedily selects a SNP, which covers most uncovered grids in the ith row at each iteration. In other words, while working on the ith row, a SNP is selected if its reformulated set S' maximizes $|S' \cap R_i|$, where R_i is the set of uncovered grids at the ith row.

Figure 77.4 illustrates an example for this algorithm to tolerate one missing tag SNP (i.e., $m = 1$). The SNPs S_1, S_4, S_2, and S_3 are selected in order. When all grids in this table are covered, each pair of patterns is distinguished by $(m+1)$ SNPs in the corresponding column. Thus, the SNPs in this table are the robust tag SNPs which allows at most m SNPs genotyped as missing data. The pseudocode of this greedy algorithm is given below.

Algorithm: First-Greedy-Algorithm(C, P, m)

```
1   $R_i \leftarrow P, \forall i \in [1, m+1]$
2   $C' \leftarrow \phi$
3   for $i = 1$ to $m+1$ do
4       while $R_i \neq \phi$ do
5           select and remove a SNP $S$ from $C$ that maximizes $|S' \cap R_i|$
6           $C' \leftarrow C' \cup S$
7           $j \leftarrow i$
8           while $S' \neq \phi$ and $j \leq m+1$ do
9               $S_{tmp} \leftarrow S' \cap R_j$
10              $R_j \leftarrow R_j - S_{tmp}$
11              $S' \leftarrow S' - S_{tmp}$
12              $j \leftarrow j+1$
13          endwhile
14      endwhile
15  endfor
16  return $C'$
```

The time complexity of this algorithm is analyzed as follows. At Line 4, the number of iterations of the intermediate loop is bounded by $|R_i| \leq |P|$. Within the loop body (Lines 5–13), Line 5 takes



C'

$S_1' = \{(1,3), (1,4), (2,3), (2,4)\}$

$S_4' = \{(1,2), (1,4), (2,3), (3,4)\}$

$S_2' = \{(1,2), (1,3), (1,4)\}$

$S_3' = \{(1,4), (2,4), (3,4)\}$

P

(1, 2)	**(1, 3)**	**(1, 4)**	**(2, 3)**	**(2, 4)**	**(3, 4)**
1/2	1/4	1/4	1/4	1/4	1/2
1/2	1/2	0	0	1/2	1/2

FIGURE 77.5 The cost C_j^i of each grid for the first greedy algorithm.

$O(|C||P|)$ because we need to check all SNPs in C and examine the uncovered grids of R_i. The inner loop (Lines 8–13) takes only $O(|S'|)$. Thus, the entire program runs in $O(m|C||P|^2)$.

We now show the number of SNPs in the solution C' returned by the first greedy algorithm is not extremely large compared to the ones in the optimal solution C^*. Suppose the algorithm selects the kth SNP when working on the ith row. Let $|S_k^c|$ be the number of grids in the ith row covered by the kth selected SNP (i.e., $|S_k^c| = |S' \cap R_i|$; see Line 5 in FIRST-GREEDY-ALGORITHM). For example (see Figure 77.4), $S_2^c = 2$ since the second selected SNP (i.e., S_4) covers two grids in the first row. We incur 1 unit of cost to each selected SNP, and spread this cost among the grids in S_k^c [15]. In other words, each grid at the ith row and jth column is assigned a cost C_j^i (see Figure 77.5), where

$$C_j^i = \begin{cases} \frac{1}{|S_k^c|} & \text{if the algorithm selects the } k\text{th SNP when covering the } i\text{th row} \\ 0 & \text{otherwise} \end{cases}$$

Since each selected SNP is assigned 1 unit of cost, the sum of C_j^i for each grid in the table is equal to $|C'|$, that is,

$$|C'| = \sum_{i=1}^{m+1} \sum_{j=1}^{\frac{K(K-1)}{2}} C_j^i \qquad (77.1)$$

Let R_k^i be the number of uncovered grids in the ith row before the kth iteration (i.e., $(k-1)$ SNPs have been selected by the algorithm). For example (see Figure 77.5), $R_2^1 = 2$ since two grids in the first row are still uncovered before the second SNP is selected. Define C_i' as the set of iterations used by the algorithm when working on the ith row. For example (see Figure 77.5), $C_2' = \{3, 4\}$, since this algorithm works on the second row in the third and fourth iterations. We can rewrite Eq. (77.1) as

$$\sum_{i=1}^{m+1} \sum_{j=1}^{\frac{K(K-1)}{2}} C_j^i = \sum_{i=1}^{m+1} \sum_{k \in C_i'} (R_{k-1}^i - R_k^i) \frac{1}{|S_k^c|} \qquad (77.2)$$

Lemma 77.3

The kth selected SNP has $|S_k^c| \geq \frac{R_{k-1}^i}{|C^*|}$.

Proof

Suppose the algorithm is working on the ith row at the beginning of the kth iteration. Let C_k^* be the set of SNPs in C^* (the optimal solution) that has been selected by the algorithm before the kth iteration, and the set of remaining SNPs in C^* be $C_{\bar{k}}^*$. We claim that there exists a SNP in $C_{\bar{k}}^*$, which can cover at least $\frac{R_k^i}{|C_{\bar{k}}^*|}$ grids in the ith row. Otherwise (i.e., each SNP in $C_{\bar{k}}^*$ covers less than $\frac{R_k^i}{|C_{\bar{k}}^*|}$ grids), all SNPs in $C_{\bar{k}}^*$ will cover less than $(\frac{R_k^i}{|C_{\bar{k}}^*|} \times |C_{\bar{k}}^*| = R_k^i)$ grids in the ith row. But since $C_k^* \cup C_{\bar{k}}^* = C^*$, this implies that C^* cannot cover all grids in R_k^i, which is a contradiction. Because all SNPs in $C_{\bar{k}}^*$ are candidates to the greedy algorithm, the kth selected SNP must cover at least $\frac{R_k^i}{|C_{\bar{k}}^*|}$ grids in the ith row, which implies $|S_k^c| \geq \frac{R_{k-1}^i}{|C^*|}$ since $|C^*| \geq |C_{\bar{k}}^*|$ and $|R_k^i| \leq |R_{k-1}^i|$. □

Theorem 77.2

The first greedy algorithm gives a solution of $(m+1) \ln \frac{K(K-1)}{2}$ approximation.

Proof

Define the dth harmonic number as $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$. By Eq. (77.2) and Lemma 77.3,

$$
\begin{aligned}
\sum_{i=1}^{m+1} \sum_{j=1}^{\frac{K(K-1)}{2}} C_j^i &= \sum_{i=1}^{m+1} \sum_{k \in C_i'} \left(R_{k-1}^i - R_k^i\right) \frac{1}{|S_k^c|} \le \sum_{i=1}^{m+1} \sum_{k \in C_i'} \left(R_{k-1}^i - R_k^i\right) \frac{|C^*|}{R_{k-1}^i} \\
&= \sum_{i=1}^{m+1} \sum_{k \in C_i'} \left(\sum_{l=R_k^i+1}^{R_{k-1}^i} \frac{|C^*|}{R_{k-1}^i} \right) \\
&\le |C^*| \sum_{i=1}^{m+1} \sum_{k \in C_i'} \sum_{l=R_k^i+1}^{R_{k-1}^i} \frac{1}{l} \qquad \left(l \le R_{k-1}^i\right) \\
&= |C^*| \sum_{i=1}^{m+1} \sum_{k \in C_i'} \left(\sum_{l=1}^{R_{k-1}^i} \frac{1}{l} - \sum_{l=1}^{R_k^i} \frac{1}{l} \right) \\
&\le |C^*| \sum_{i=1}^{m+1} \sum_{k \in C_i'} \left(H\left(R_{k-1}^i\right) - H\left(R_k^i\right) \right) \\
&\le |C^*| \sum_{i=1}^{m+1} \left(H\left(R_0^i\right) - H\left(R_{|C_i'|}^i\right) \right) \\
&\le |C^*|(m+1)\max\{H(R_0^i)\} \qquad (R_{|C_i'|}^i = 0 \text{ and } H(0) = 0\ ) \\
&\le |C^*|(m+1)\ln|P| \qquad\qquad (H(R_0^i) \le H(|P|))
\end{aligned}
\tag{77.3}
$$

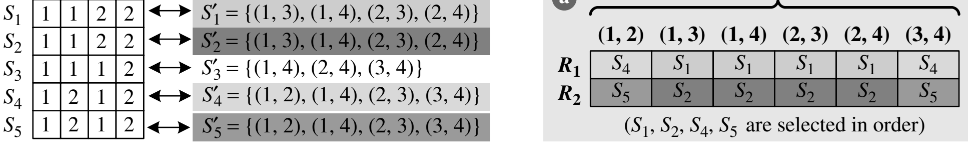
By Eq. (77.1) and Eq. (77.3), we get

$$
\frac{|C'|}{|C^*|} \le (m+1)\ln|P| = (m+1)\ln\frac{K(K-1)}{2} \qquad \square
$$

77.2.2 The Second Greedy Algorithm

This section describes the second greedy algorithm which returns a better solution than the one the first greedy algorithm generates. Let R_i be the set of uncovered grids at the ith row. Unlike the row-by-row manner of the first greedy algorithm, this algorithm greedily selects a SNP that covers most uncovered grids in the table (i.e., its reformulated set S' maximizing $|S' \cap (R_1 \cup \cdots \cup R_{m+1})|$). Let T be the collection of R_i (i.e., T is the set of all uncovered grids in the table). If the grids in the ith row are all covered (i.e., $R_i = \phi$), R_i is removed from T. This algorithm runs until $T = \phi$ (i.e., all grids in the table are covered).

Figure 77.6 illustrates an example for this algorithm with m set to 1. The SNPs S_1, S_2, S_4, and S_5 are selected in order. Since this algorithm runs until all grids are covered, the set of SNPs in this table is able to tolerate m missing tag SNPs. The pseudocode of this algorithm is given below.



	P_1	P_2	P_3	P_4	
S_1	1	1	2	2	⟷ $S_1' = \{(1,3), (1,4), (2,3), (2,4)\}$
S_2	1	1	2	2	⟷ $S_2' = \{(1,3), (1,4), (2,3), (2,4)\}$
S_3	1	1	1	2	⟷ $S_3' = \{(1,4), (2,4), (3,4)\}$
S_4	1	2	1	2	⟷ $S_4' = \{(1,2), (1,4), (2,3), (3,4)\}$
S_5	1	2	1	2	⟷ $S_5' = \{(1,2), (1,4), (2,3), (3,4)\}$

a

P

	(1, 2)	**(1, 3)**	**(1, 4)**	**(2, 3)**	**(2, 4)**	**(3, 4)**
$\boldsymbol{R_1}$	S_4	S_1	S_1	S_1	S_1	S_4
$\boldsymbol{R_2}$	S_5	S_2	S_2	S_2	S_2	S_5

(S_1, S_2, S_4, S_5 are selected in order)

FIGURE 77.6 The SNPs S_1, S_2, S_4, and S_5 are selected by the second greedy algorithm. (a) The table that stores each selected SNP.

Algorithm: SECOND-GREEDY-ALGORITHM(C, P, m)

```
1   $R_i \leftarrow P, \forall i \in [1, m+1]$
2   $T \leftarrow \{R_1, R_2, \ldots, R_{m+1}\}$
3   $C' \leftarrow \phi$
4   while $T \neq \phi$ do
5       select and remove a SNP $S$ from $C$ that maximizes $|S' \cap (R_1 \cup \cdots \cup R_{m+1})|$
6       $C' \leftarrow C' \cup S$
7       for each $R_i \in T$ and $S' \neq \phi$ do
8           $S_{tmp} \leftarrow S' \cap R_i$
9           $R_i \leftarrow R_i - S_{tmp}$
10          $S' \leftarrow S' - S_{tmp}$
11          if $R_i = \phi$ then $T \leftarrow T - R_i$
12      endfor
13  endwhile
14  return $C'$
```

The time complexity of this algorithm is analyzed as follows. At Line 4, the number of iterations of the loop is bounded by $O(|T|) = O(m|P|)$. Within the loop, Line 5 takes $O(|C||P|)$ time because we need to check each SNP in C and examine if it can cover any uncovered grid in each column. The inner loop (Lines 7–12) is bounded by $O(|S'|) < O(|P|)$. Thus, the running time of this program is $O(m|C||P|^2)$.

We now evaluate the solution returned by the second greedy algorithm. Let C' and C^* be the set of SNPs selected by this algorithm and the optimal solution, respectively. Let $|S_k^c|$ be the number of grids in the table covered by the kth selected SNP. For example (see Figure 77.6), $|S_2^c| = 4$ since the second selected SNP (i.e., S_2) covers four grids in the table. Define T_k as the number of uncovered grids in the table before the kth iteration. We have the following lemma similar to Lemma 77.3.

Lemma 77.4

The kth selected SNP has $|S_k^c| \geq \frac{T_{k-1}}{|C^*|}$.

Proof

The proof is similar to that of Lemma 77.3. Let C_k^* be the set of remaining SNPs in C^*, which has not been selected before the kth iteration. We claim that there exists a SNP in C_k^*, which can cover at least $\frac{T_k}{|C_k^*|}$ grids in the table. Otherwise, we can get the same contradiction (i.e., C^* fails to cover all grids) as in Lemma 77.3. Since $|C^*| \geq |C_k^*|$ and $T_{k-1} \leq T_k$, we have $|S_k^c| \geq \frac{T_{k-1}}{|C^*|}$. □

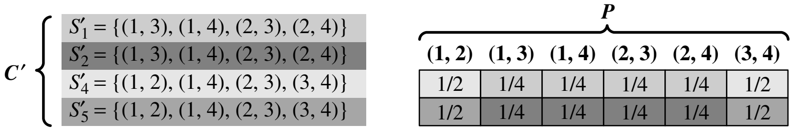
Theorem 77.3

The second greedy algorithm gives a solution of $\ln((m+1)\frac{K(K-1)}{2})$ *approximation.*

Proof

Each grid at the ith row and jth column is assigned a cost $C_j^i = \frac{1}{|S_k^c|}$ (see Figure 77.7) if it is covered by the kth selected SNP. The sum of C_j^i for each grid is

$$\begin{aligned}
|C'| = \sum_{i=1}^{m+1} \sum_{j=1}^{\frac{K(K-1)}{2}} C_j^i &= \sum_{k=1}^{|C'|} (T_{k-1} - T_k) \frac{1}{|S_k^c|} && \text{(see Eq. (77.1) and Eq. (77.2))} \\
&\leq \sum_{k=1}^{|C'|} (T_{k-1} - T_k) \frac{|C^*|}{T_{k-1}} && \text{(by Lemma 77.4)} \\
&\leq |C^*|(H(T_0) - H(T_{|C'|})) && \text{(see the proof in Theorem 77.2)} \\
&\leq |C^*| \ln((m+1)|P|) && (77.4)
\end{aligned}$$



C'

$S'_1 = \{(1, 3), (1, 4), (2, 3), (2, 4)\}$

$S'_2 = \{(1, 3), (1, 4), (2, 3), (2, 4)\}$

$S'_4 = \{(1, 2), (1, 4), (2, 3), (3, 4)\}$

$S'_5 = \{(1, 2), (1, 4), (2, 3), (3, 4)\}$

P

(1, 2)	(1, 3)	(1, 4)	(2, 3)	(2, 4)	(3, 4)
1/2	1/4	1/4	1/4	1/4	1/2
1/2	1/4	1/4	1/4	1/4	1/2

FIGURE 77.7 The cost C_j^i of each grid for the second greedy algorithm.

By Eq. (77.4), we have

$$\frac{|C'|}{|C^*|} \le \ln((m+1)|P|) = \ln((m+1)\frac{K(K-1)}{2})$$

□

77.2.3 The Iterative LP-Relaxation Algorithm

In practice, a probabilistic approach is sometimes more useful since the randomization can explore different solutions. In this section, we reformulate the MRTS problem to an *Integer Programming* (IP) problem.

Based on the IP problem, we propose an iterative LP-relaxation algorithm. The iterative LP-relaxation algorithm is described below.

Step 1. Given a haplotype block containing N SNPs and K haplotype patterns. Let $\{x_1, x_2, \ldots, x_N\}$ be the set of integer variables for the N SNPs, where $x_k = 1$ if the SNP S_k is selected and $x_k = 0$ otherwise. Define $D(P_i, P_j)$ as the set of SNPs which are able to distinguish P_i and P_j patterns. By Lemma 77.1, to allow at most m SNPs genotyped as missing data, each pair of patterns must be distinguished by at least $(m+1)$ SNPs. Therefore, for each set $D(P_i, P_j)$, at least $(m+1)$ SNPs have to be selected to distinguish P_i and P_j patterns. As a consequence, the MRTS problem can be formulated as the following IP problem:

$$\begin{aligned} &\textbf{Minimize} && \sum_{k=1}^{N} x_k \\ &\textbf{Subject to} && \sum_{k\in D(P_i,P_j)} x_k \ge m+1, \quad \text{for all } 1 \le i < j \le K \\ &&& x_k = 0 \text{ or } 1 \end{aligned} \tag{77.5}$$

Step 2. Since solving the IP problem is NP-hard [13], we relax the integer constraint of x_k, and the IP problem becomes a LP problem defined as follows:

$$\begin{aligned} &\textbf{Minimize} && \sum_{k=1}^{N} y_k \\ &\textbf{Subject to} && \sum_{k\in D(P_i,P_j)} y_k \ge m+1, \quad \text{for all } 1 \le i < j \le K \\ &&& 0 \le y_k \le 1 \end{aligned} \tag{77.6}$$

The above LP problem can be solved by efficient algorithms such as the interior point method [16,17].

Step 3. Let $\{y_1, y_2, ..., y_N\}$ be the set of linear solutions obtained from Eq. (77.6), where $0 \le y_k \le 1$. We assign 0 or 1 to x_k by the following randomized rounding method:

$$\text{Assign} \begin{cases} x_k = 1 \text{ with probability } y_k \\ x_k = 0 \text{ with probability } 1 - y_k \end{cases}$$

Step 4. The randomized rounding method may invalidate some of the inequalities in Eq. (77.5). Thus, we repeat steps 1–3 for those unsatisfied inequalities until all of them are satisfied. Finally, when all inequalities in Eq. (77.5) are satisfied, we construct a final solution by the following rule:

$$\text{Assign} \begin{cases} x_k = 1 & \text{if } x_k \text{ is assigned to 1 in any one of the iterations} \\ x_k = 0 & \text{otherwise} \end{cases}$$

We now evaluate the solution returned by the iterative LP-relaxation algorithm. The selection of each SNP is considered as a *Bernoulli* random variable x_k taking values 1 (or 0) with probability y_k (or $1 - y_k$). Let $X_{i,j}$ be the sum of random variables in one inequality of Eq. (77.5), that is,

$$X_{i,j} = \sum_{k \in D\{P_i, P_j\}} x_k$$

By Eq. (77.6), the expected value of $X_{i,j}$ (after randomized rounding) is

$$\begin{aligned} E[X_{i,j}] &= \sum_{k \in D\{P_i, P_j\}} E[x_k] = \sum_{k \in D\{P_i, P_j\}} y_k \\ &\geq m + 1 \end{aligned} \tag{77.7}$$

Lemma 77.5

The probability that an inequality in Eq. (77.5) *is not satisfied after randomized rounding is less than* $e^{-\frac{1}{2(m+1)}}$.

Proof

The probability that an inequality in Eq. (77.5) is not satisfied is $P[X_{i,j} < m + 1] = P[X_{i,j} \leq m]$. By the *Chernoff* bound (i.e., $P[X \leq (1 - \theta)E[X]] \leq e^{-\frac{\theta^2 E[X]}{2}}$), we have

$$P[X_{i,j} \leq m] \leq e^{-\frac{(E[X_{i,j}]-m)^2}{2E[X_{i,j}]}} \tag{77.8}$$

By Eq. (77.7), we know $E[X_{i,j}] \geq m+1$. Since the right-hand side of Eq. (77.8) decreases when $E[X_{i,j}] > m$, we can replace $E[X_{i,j}]$ with $(m + 1)$ to obtain an upper bound, that is,

$$\begin{aligned} P[X_{i,j} \leq m] &\leq e^{-\frac{(E[X_{i,j}]-m)^2}{2E[X_{i,j}]}} \leq e^{-\frac{(m+1-m)^2}{2(m+1)}} \\ &\leq e^{-\frac{1}{2(m+1)}} \end{aligned}$$

□

Theorem 77.4

The iterative LP-relaxation algorithm gives a solution of $O(m \ln K)$ *approximation.*

Proof

Suppose this algorithm runs for t iterations. The probability that all $\frac{K(K-1)}{2}$ inequalities in Eq. (77.5) are satisfied after t iterations is

$$\begin{aligned} (1 - (e^{-1/2(m+1)})^t)^{\frac{K(K-1)}{2}} &= (1 - e^{-t/2(m+1)})^{\frac{K(K-1)}{2}} \\ &\approx e^{-\frac{K(K-1)}{2} e^{-t/2(m+1)}} \end{aligned}$$

When $t = 2(m + 1) \ln \frac{K(K-1)}{2}$, the algorithm stops and returns a solution with probability e^{-1}. Define $OPT(IP)$ and $OPT(LP)$ as the optimal solutions of the IP problem and the LP problem, respectively. Since the solution space of LP includes that of IP,

$$OPT(LP) \leq OPT(IP)$$

Let the set of solutions returned in t iterations be $\{Z_1, Z_2, \ldots, Z_t\}$.

$$E[Z_1] = E\left[\sum_{k=1}^{N} x_k\right] = \sum_{k=1}^{N} y_k = OPT(LP)$$

Note that we repeat this algorithm only for those unsatisfied inequalities. Thus, $E[Z_1] \geq E[Z_2] \geq \cdots \geq E[Z_t]$. Let x_p denote the final solution obtained in step 4. The expected final solution is

$$\begin{aligned}
E\left[\sum_{p=1}^{N} x_p\right] &\leq E\left[\sum_{p=1}^{t} Z_p\right] \\
&\leq t \times E[Z_1] \\
&\leq t \times OPT(LP) \\
&\leq 2(m+1)\ln\frac{K(K-1)}{2} \times OPT(IP) \\
&= O(m \ln K) \times OPT(IP)
\end{aligned}$$

With a high probability, the iterative LP-relaxation algorithm stops after $O(m \ln K)$ iterations and finds a solution of $O(m \ln K)$ approximation. □

77.3 Finding Auxiliary Tag SNPs

This section describes an algorithm for finding auxiliary tag SNPs assuming that robust tag SNPs have been computed in advance. Given a haplotype block M_h containing N SNPs and K haplotypes, we define $C_{tag} \subseteq C$ as the set of tag SNPs obtained from a haplotype sample and some SNPs in C_{tag} are missing. This haplotype sample may be identified ambiguously due to the lack of missing SNPs. We wish to find the minimum number of auxiliary tag SNPs from the remaining SNPs to resolve the ambiguity. A formal definition of this problem is given below.

Problem: Minimum Auxiliary Tag SNPs

Input. An $N \times K$ matrix M_h, and a set of SNPs C_{tag} genotyped from a sample with missing data.

Output. The minimum subset of SNPs $C_{aux} \subseteq C - C_{tag}$ such that each pair of ambiguous patterns can be distinguished by SNPs in C_{aux}.

The following theorem shows the NP-hardness of the Minimum Auxiliary Tag SNPs (MATS) problem.

Theorem 77.5

The MATS problem is NP-hard.

Proof

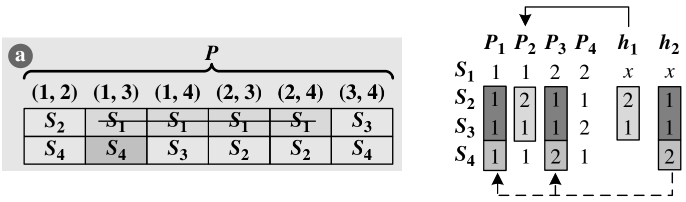
Consider that all SNPs in C_{tag} are genotyped as missing data. This special case of the MATS problem is just like finding another set of tag SNPs from $C - C_{tag}$ to distinguish those K patterns, which is already known as NP-hard [11]. □

Although the MATS problem is NP-hard, we show that auxiliary tag SNPs can be found efficiently when robust tag SNPs have been computed in advance. Without loss of generality, assume that these robust tag SNPs are stored in an $(m+1) \times |P|$ table T_r (see Figure 77.8 [a]).

Step 1. The patterns that match the haplotype sample are stored into a set A. For example (see Figure 77.8), if we genotype SNPs S_1, S_2, and S_3 for the sample h_2 and the SNP S_1 is missing, patterns P_1 and P_3 both match h_2. Thus, $A = \{P_1, P_3\}$.

Step 2. If $|A|$=1, the sample is identified unambiguously and we are done (e.g., h_1 in Figure 77.8). If $|A| > 1$ (e.g., h_2), for each pair of ambiguous patterns in A (e.g., P_1 and P_3), traverse the corresponding column in T_r, find the next unused SNP (e.g., S_4), and add the SNP to C_{aux}. As a result, the SNPs in C_{aux} can distinguish each pair of ambiguous patterns, which are the auxiliary tag SNPs for the haplotype sample.

The worst case of this algorithm is that all SNPs in C_{tag} are genotyped as missing data, and we need to traverse each column in T_r. Thus, the running time of this algorithm is $O(|T_r|) = O(m|P|)$.



(1, 2)	(1, 3)	(1, 4)	(2, 3)	(2, 4)	(3, 4)
S_2	~~S_1~~	~~S_1~~	~~S_1~~	~~S_1~~	S_3
S_4	S_4	S_3	S_2	S_2	S_4

	P_1	P_2	P_3	P_4	h_1	h_2
S_1	1	1	2	2	x	x
S_2	1	2	1	1	2	1
S_3	1	1	1	2	1	1
S_4	1	1	2	1		2

FIGURE 77.8 An example to find the auxiliary tag SNPs. The SNP S_1 is genotyped as missing data and SNP S_4 is the auxiliary tag SNP for h_2. (a) The table that stores the set of robust tag SNPs.

77.4 Experimental Results

We have implemented the first and second greedy algorithms in JAVA. The LP-relaxation algorithm has been implemented in Perl, where the LP problem is solved via a program called "lp_solve" [16]. The LP-relaxation algorithm is a randomized method. Thus, this program is repeated for 10 times to explore different solutions and the best solution among them is chosen as the output. To compare the solutions (and efficiency) returned by our algorithms with the optimal solution, we also implement a brute force program in JAVA (referred to as "OPT") which enumerates all possible solutions to find the optimal solution. The proposed algorithms along with the brute force program are tested on a variety of simulated and biological data.

77.4.1 Results on Simulated Data

We first generate 100 data sets containing short haplotypes. Each data set consists of 10 haplotypes with 20 SNPs. These haplotypes are created by randomly assigning the major or minor alleles at each SNP locus. Let m be the number of missing SNPs allowed and S_a be the average number of robust tag SNPs over 100 data sets. Figure 77.9(a) plots S_a with respect to m (roughly corresponding to SNP missing rates from 0% to 33%). When $m = 0$, all programs find the same number of SNPs as the optimal solution. The iterative LP-relaxation algorithm slightly outperforms others as m increases. When $m > 6$, more than 20 SNPs are required to tolerate missing data. Thus, no data sets contain enough SNPs for solutions.

We then generate 100 data sets containing long haplotypes. Each data set is composed of 10 haplotypes with 40 SNPs. Figure 77.9(b) illustrates the experimental results on these long data sets (corresponding to SNP missing rates from 0 to 37%). The optimal solutions for $m > 1$ cannot be computed in one day and are not shown in this figure. It is because the number of possible solutions in long data sets is too large

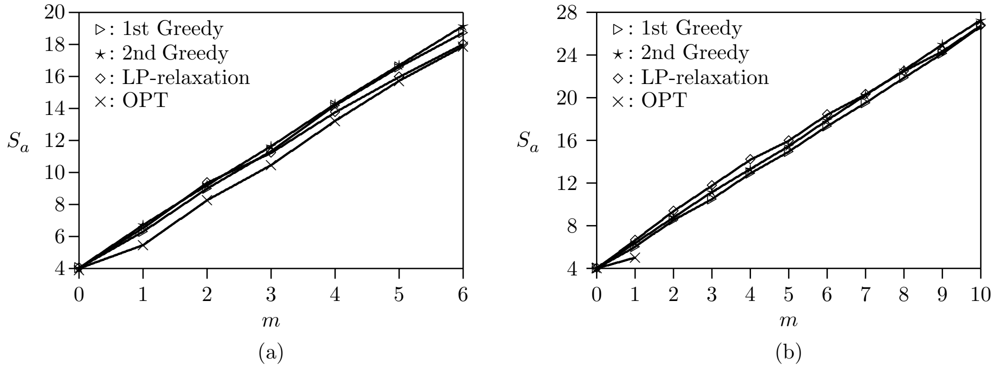


FIGURE 77.9 Experimental results on random data. (a) Results from data sets containing 10 haplotypes and 20 SNPs. (b) Results from data sets containing 10 haplotypes and 40 SNPs.

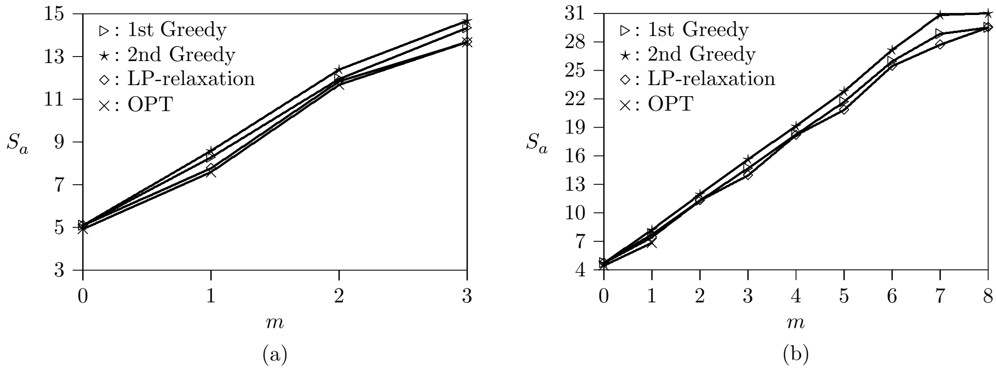


FIGURE 77.10 Experimental results on Hudson's data. (a) Results from data sets containing 10 haplotypes and 20 SNPs. (b) Results from data sets containing 10 haplotypes and 40 SNPs.

to enumerate. However, both greedy and iterative LP-relaxation algorithms run in polynomial time and always output a solution efficiently. In this experiment, both greedy algorithms slightly outperform the iterative LP-relaxation algorithm. In addition, the number of SNPs allowed for missing data is larger than those in short data sets. For example, when $m = 10$, all programs output less than 28 SNPs. The remaining SNPs in each data set are still enough to tolerate more missing SNPs.

Hudson [18] provides a program that can simulate a set of haplotypes under the assumption of neutral evolution and uniformly distributed recombination rate using the coalescent model. We use Hudson's program to generate 100 short data sets with 10 haplotypes and 20 SNPs and 100 long data sets with 10 haplotypes and 40 SNPs. Figure 77.10(a) shows the experimental results on Hudson's short data sets (corresponding to SNP missing rates from 0 to 23%). The number of missing SNPs allowed are less than that of random data. It is because Hudson's program generates coalescent haplotypes that are similar to each other. As a result, many SNPs cannot be used to distinguish those haplotypes and the amount of tag SNPs is inadequate to tolerate larger missing SNPs. In this experiment, we observe that the iterative LP-relaxation algorithm finds solutions quite close to the optimal solutions and slightly outperforms the other two algorithms.

Figure 77.10(b) illustrates the experimental results on long data sets generated by Hudson's program (corresponding to SNP missing rates from 0 to 29%). The optimal solutions for $m > 1$ again cannot be computed in one day. In this experiment, the performance of the first greedy and iterative LP-relaxation algorithms are similar, and they slightly outperform the second greedy algorithm as m becomes large.

77.4.2 Results on Biological Data

We test these programs on public haplotype data of human Chromosome 21 released by Patil et al. [5]. Patil's data includes 20 haplotypes of 24,047 SNPs spanning over about 32.4MB. Based on the 4,135 haplotype blocks partitioned by Patil et al., we apply all programs to find the robust tag SNPs in each block. Figure 77.11(a) shows the experimental results on these 4,135 blocks. Because there are many long blocks in Patil's data (e.g., more than one hundred SNPs), the optimal solution for $m > 1$ cannot be computed in one day. However, some short blocks may not have solutions for larger m because of insufficient number of SNPs in the block. As a consequence, S_a here stands for the average number of robust tag SNPs over those blocks containing solutions. In this experiment, all algorithms find similar number of robust tag SNPs. We observe that the number of robust tag SNPs required in Patil's data is less than those in simulated data. For example, when $m = 8$, all algorithms find less than 12 SNPs in Patil's data, and they find about 28 SNPs in random data and 31 SNPs in Hudson's data. This result implies that genotyping additional tag SNPs to tolerate missing data is more cost-effective on biological data than on simulated data.

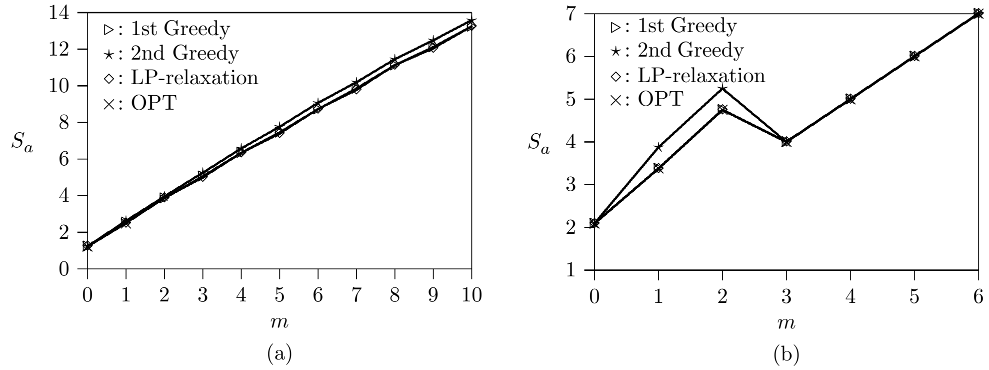


FIGURE 77.11 Experimental results on biological data. (a) Results from Patil's chromosome 21 data. (b) results from Daly's chromosome 5q31 data.

Daly et al. [4] studied a 500 kb region on human Chromosome 5q31, which may contain a genetic variant related to the Crohn disease. By genotyping 103 SNPs with minor allele frequency at least 5%, they partition this chromosomal region into 11 haplotype blocks. Figure 77.11(b) illustrates the experimental results on these 11 blocks. Because the blocks partitioned by Daly et al. are very short (e.g., most blocks contain less than 12 SNPs), the optimal solution is still computable. The solutions found by each algorithm is almost the same as optimal solutions. Note that the number of blocks (containing solutions) decreases as m increases. When m increases to 3, some blocks that do not have enough SNPs for a solution are discarded. The remaining block require only 4 SNPs and S_a thus drops down to 4.

77.4.3 Discussion

In terms of efficiency, the first and second greedy algorithms are faster than the LP-relaxation algorithm. The greedy algorithms usually return a solution in seconds and the LP-relaxation algorithm requires about half minute for a solution. It is because the running time of LP-relaxation algorithm is bounded by the time of solving the LP problem. Furthermore, this LP-relaxation algorithm is repeated and rounded for 10 times to explore 10 different solutions. The brute force program for searching the optimal solution is apparently slower than the others (e.g., taking hours for a solution). The optimal solution usually cannot be found in 24 hours if the size of the block becomes large. When m increases, each block requires more tag SNPs to tolerate more missing SNPs, and the brute force program requires longer execution time to find the optimal solution.

Assuming no missing data (i.e., $m = 0$), we now compare the solutions found by each algorithm with the optimal solution. Table 77.1 lists the numbers of total tag SNPs found by each algorithm in previous experiments. In the experiments on random and Daly's data, the solution found by each algorithm is as good as the optimal solution. In the experiments on Hudson's and Patil's data, these algorithms still find solutions quite close to the optimal solution. For example, the approximation ratios of these algorithms are only $\frac{472}{443} \approx 1.07$ and $\frac{4657}{4595} \approx 1.01$, respectively.

We then analyze the genotyping cost that can be saved by using tag SNPs. In Table 77.1, the ratio of tag SNPs to total SNPs in each data set is shown in parentheses. The experimental results indicate that the cost of genotyping tag SNPs is much lower than that of genotyping all SNPs in a block. For example, in Patil's data, we only need to genotype about 19% of tag SNPs in each block, which saves about 81% genotyping cost. The genotyping cost saved by using tag SNPs is especially significant in long haplotype blocks. For example, in random and Hudson's long data sets, the saved genotyping cost can be as high as 90%.

Finally, we compute the cost of genotyping extra tag SNPs for tolerating missing data. The biological data sets (i.e., Patil's and Daly's data) have blocks in different sizes. Some of them may not have solutions for larger values of m. Therefore, we only consider random and Hudson's 100 data sets in the same size.

TABLE 77.1 The Number of Total Tag SNPs Found by Each Algorithm. The Ratio of Tag SNPs to Total SNPs Is Shown in Parentheses

	Random Data		Hudson's Data		Patil's Data	Daly's Data
Total blocks	100	100	100	100	4135	11
Total SNPs	2000	4000	2000	4000	24047	103
1st Greedy	400 (20%)	400 (10%)	509 (25.5%)	472 (11.8%)	4610 (19.2%)	23 (22.3%)
2nd Greedy	400 (20%)	400 (10%)	509 (25.5%)	472 (11.8%)	4610 (19.2%)	23 (22.3%)
LP-relaxation	400 (20%)	400 (10%)	509 (25.5%)	471 (11.8%)	4657 (19.4%)	23 (22.3%)
OPT	400 (20%)	400 (10%)	492 (24.6%)	443 (11.1%)	4595 (19.1%)	23 (22.3%)

TABLE 77.2 The Number of Extra Tag SNPs Required to Tolerate Missing Data. The Ratio of Extra Tag SNPs to Total SNPs Is Shown in Parentheses

m		1	2	3	4	5
Random	1st Greedy	200 (5.0%)	451 (11.3%)	647 (16.2%)	889 (22.2%)	1092 (27.3%)
data	2nd Greedy	237 (5.9%)	477 (11.9%)	714 (17.9%)	930 (23.3%)	1144 (28.6%)
(4000 SNPs)	LP-relaxation	262 (6.6%)	535 (13.4%)	774 (19.4%)	1018 (25.5%)	1194 (29.9%)
Hudson's	1st Greedy	299 (7.5%)	656 (16.4%)	995 (24.9%)	1351 (33.8%)	1695 (42.4%)
data	2nd Greedy	347 (8.7%)	723 (18.1%)	1091 (27.3%)	1439 (36.0%)	1806 (45.2%)
(4000 SNPs)	LP-relaxation	269 (6.7%)	657 (16.4%)	921 (23.0%)	1344 (33.6%)	1609 (40.2%)

Each data set contains 10 haplotypes with 40 SNPs and has solutions for m from 1 to 5. Table 77.2 lists the number of extra tag SNPs used by each algorithm. When m increases to 5, the extra genotyping cost is less than 30% for all algorithms on random data. In contrast, the extra genotyping cost is higher on Hudson's data because of the coalescent haplotypes. However, in comparison with genotyping all SNPs, the extra genotyping cost is still less than 50% and is thus cost-effective.

77.5 Concluding Remarks

In this chapter, we show there exists a set of robust tag SNPs, which is able to tolerate a certain number of missing data. Our study indicates that robust tag SNPs is more practical than the minimum tag SNPs if we cannot avoid the occurrence of missing data. We describe two greedy and one LP-relaxation approximation algorithms for finding robust tag SNPs. Our experimental results and theoretical analysis show that these algorithms are not only efficient but the solutions found are also close to the optimal solution. In terms of genotyping cost, we observe that the genotyping cost saved by using tag SNPs can be as high as 90%, and genotyping extra tag SNPs to tolerate missing data is still cost-effective. One future direction is to assign weights to different types of SNPs (e.g., SNPs in coding or noncoding regions), and design algorithms for the selection of weighted tag SNPs.

References

[1] Helmuth, L., Genome research: map of the human genome 3.0, *Science*, 293(5530), 583, 2001.

[2] Hinds, D. A., Stuve, L. L., Nilsen, G. B., Halperin, E., Eskin, E., Ballinger, D. G., Frazer, K. A., and Cox, D. R., Whole-genome patterns of common DNA variation in three human populations, *Science*, 307, 1072, 2005.

[3] Bafna, V. and Bansal, V., Improved recombination lower bounds for haplotype data, *Proc. RECOMB*, 569–584, 2005.

[4] Daly, M. J., Rioux, J. D., Schaffner, S. F., Hudson, T. J., and Lander, E. S., High-resolution haplotype structure in the human genome, *Nat. Genet.*, 29(2), 229, 2001.

[5] Patil, N., Berno, A. J., Hinds, D. A., Barrett, W. A., Doshi, J. M., Hacker, C. R., Kautzer, C. R., Lee, D. H., Marjoribanks, C., McDonough, D. P., Nguyen, B. T. N., Norris, M. C., Sheehan, J. B., Shen, N., Stern, D., Stokowski, R. P., Thomas, D. J., Trulson, M. O., Vyas, K. R., Frazer, K. A., Fodor, S. P. A., Cox, D. R., Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21, *Science*, 294, 1719, 2001.

[6] Halperin, E. and Eskin, E., Haplotype reconstruction from genotype data using imperfect phylogeny, *Bioinformatics*, 1842–1849, 2004.

[7] Zhang, K., Qin, Z. S., Liu, J. S., Chen, T., Waterman, M. S., and Sun, F., Haplotype block partition and tag SNP selection using genotype data and their applications to association studies, *Genome Res.*, 14, 908, 2004.

[8] Bafna, V., Halldórsson, B. V., Schwartz, R., Clark, A. G., and Istrail, S, Haplotypes and informative SNP selection algorithms: don't block out information, *Proc. RECOMB*, 2003, p. 19.

[9] Carlson, C. S., Eberle, M. A., Rieder, M. J., Yi, Q., Kruglyak, L., and Nickerson, D.A, Selecting a maximally informative set of single-nucleotide polymorphisms for association analyses using linkage disequilibrium, *Am. J. Hum. Genet.*, 74, 106, 2004.

[10] Halldórsson, B. V., Bafna, V., Lippert, R., Schwartz, R., Vega, F. M., Clark, A. G., and Istrail, S., Optimal haplotype block-free selection of tagging SNPs for genome-wide association studies, *Genome Res.*, 1633–1640, 2004.

[11] Zhang, K., Deng, M., Chen, T., Waterman, M. S., and Sun, F., A dynamic programming algorithm for haplotype partitioning, *Proc. Nat. Acad. Sci.*, 99(11), 7335, 2002.

[12] Zhang, K., Sun, F., Waterman, M. S., and Chen, T., Haplotype block partition with limited resources and applications to human chromosome 21 haplotype data, *Am. J. Hum. Genet.*, 73, 63, 2003.

[13] Garey, M. R. and Johnson, D. S., *Computers and Intractability*, Freeman, New York, 1979.

[14] Zhao, J. H., Lissarrague, S., Essioux, L., and Sham, P. C., GENECOUNTING: Haplotype analysis with missing genotypes, *Bioinformatics*, 18, 1694, 2002.

[15] Cormen T. H., Leiserson, C. E., Rivest, R. L., and Stein, C, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 2001.

[16] `http://www.cs.sunysb.edu/~algorith/implement/lp_solve/implement.shtml`

[17] Forsgren, A., Gill, P. E., and Wright, M. H., Interior methods for nonlinear optimization, *SIAM Rev.*, 44, 525, 2002.

[18] Hudson, R. R., Generating samples under a Wright–Fisher neutral model of genetic variation, *Bioinformatics*, 18, 337, 2002.

78

Sphere Packing and Medical Applications

Danny Z. Chen*
University of Notre Dame

Jinhui Xu†
State University of New York at Buffalo

78.1 Introduction

In this chapter, we consider the following sphere packing problem: Given a polygonal (or polyhedral) region R (called the *container* or *domain*) in two (or higher-dimensional space and an infinite *object set* $\mathcal{O}$ of "solid" unit spheres, find a sphere packing SP for R using the spheres in $\mathcal{O}$ such that (i) each sphere in SP is inside R, (ii) no two spheres in SP intersect each other in their interior, and (iii) the volume of R covered by SP (called the *density*) is maximized.

Packing is a venerable topic in mathematics. Various versions of packing problems have been studied [1–19], depending on the shapes of the domains, the types of objects, the position restrictions on the objects, the optimization criteria, etc. Originating from number theory and crystallography, sphere packing has mysterious connections with hyperbolic geometry, Lie algebras, and the Monster simple group, and finds direct applications in number theory and pure geometry [3]. It also arises in numerous applied areas such as digital communications, cryptography, numerical evaluation of integrals, physics, chemistry, biology, antenna design, X-ray tomography, and statistical analysis on spheres [3]. Packing problems in lower dimensions d-D ($d \leq 3$) with a bounded or unbounded domain R appear in manufacturing [20–37] (e.g., stock and cloth cutting, part nesting, compaction, and containment), mesh generation [38–44], VLSI layout [45], logistics [29], scheduling [20–23,30,37,46,47] management [28], operations research, and image processing [45,48].

Most packing problems exhibit substantial difficulties. Even very restricted versions (e.g., regular-shaped objects and domains in lower dimensional spaces) have been proved to be NP-hard. The 2-D problem of packing arbitrary-shaped objects in a bounded domain [34] has been shown to have very high time

*The research by D. Z. Chen was supported in part by the National Science Foundation under Grants CCR-9988468 and CCF-0515203.

†The research by J. Xu was supported in part by the National Science Foundation under a CAREER Award CCF-0546509 and by the National Institute of Health under USPHS grant 52567.

complexity (e.g., exponential in the size of the packing). For *congruent packing* (i.e., packing copies of the same object), it is known [49] that the 2-D cases of packing fixed-sized squares or disks in a simple polygon are NP-hard. Baur and Fekete [48] considered a closely related dispersion problem: Pack *k* congruent disks in a polygon such that the radius of disks is maximized. They proved that the dispersion problem cannot be approximated arbitrarily well in polynomial time unless P = NP, and gave a $\frac{2}{3}$-approximation algorithm for the L_∞ disk case with a time bound of $O(n^{38})$.

Recent interest on the sphere packing problem was motivated by medical applications in radiosurgery [50–52]. Radiosurgery is a minimally invasive surgical procedure that uses radiation to destroy tumors inside human body. Gamma Knife is a radiation system that contains 201 Cobalt-60 sources [53,54]. The gamma-rays from these sources are all focused on a common center, creating a spherical volume of high radiation dose. A key geometric problem in Gamma Knife treatment planning is to fit multiple balls into a 3-D irregular-shaped tumor [53–55]. In such applications, overlapping balls may cause overdose, and a low packing density may result in underdose and a nonuniform dose distribution. Note that Gamma Knife currently produces spheres of four different radii (4, 8, 14, and 18 mm), and hence the Gamma Knife sphere packing is in general not congruent. However, in practice, a commonly used approach is to pack larger spheres first, and then fit smaller spheres into the remaining subdomains, in the hope of reducing the total number of spheres involved and thus shortening the treatment time. Therefore, congruent sphere packing can be used as a key subroutine for such a common approach.

Much work on congruent sphere packing studies the case of packing spheres into an unbounded domain or even the whole space [3] (e.g., Mount and Silverman's algorithm [56]). There are also some results on packing congruent spheres into a bounded region. Hochbaum and Maass [45] presented a unified and powerful *shifting technique* for designing pseudopolynomial-time approximation schemes for packing congruent squares into a rectilinear polygon; but, the high time complexities (e.g., $O(n^{38})$ in Ref. [48]) associated with the resulting algorithms restrict its applicability in practice. (In radiosurgery, since the shapes of some human organs change from time to time, a treatment is expected to be planned and delivered quickly.) Graham and Lubachevsky [57,58] considered the problems of packing k L_2 disks into an equilateral triangle or square to maximize the radius of disks, producing a number of best known packings for different constants *k*. Friedman [59] obtained many best known solutions for packing *k* unit squares into the smallest square. A common feature of this type of algorithms is to transform a packing problem into some nonlinear optimization problems, and resort to available optimization software to generate packings [51,52,60]. In general, such an approach guarantees neither a fast running time nor a provably good quality of solutions, and it works well only for small problem sizes and regular-shaped domains.

To reduce the running time yet achieve a dense packing, a common idea is to let the objects form a certain lattice or double lattice. A number of results were given on lattice packing of congruent objects in the whole (especially high-dimensional) space [3]. For a bounded rectangular 2-D domain, Milenkovic [36] adopted a method that first finds the densest translational lattice packing for a set of polygonal objects in the whole plane, and then uses some heuristics to extract the actual packing.

Chen et al. [50] present a very efficient scheme, called *pack-and-shake*, for packing congruent spheres in an irregular-shaped 2-D or 3-D bounded domain. Their scheme consists of three phases. In the first phase, the d-D ($d = 2,3$) irregular-shaped domain R is partitioned into some convex cells. The set of cells defines a dual graph G_D (each vertex v of G_D is for a cell $C(v)$, and an edge connects two vertices if their cells share a $(d-1)$-D face). In the second phase, the algorithm repeats the following *trimming and packing* process until $G_D = \emptyset$: Remove the lowest degree vertex v from G_D and pack the cell $C(v)$. In the third phase, a *shake* procedure is applied to globally adjust the packing to obtain a denser one.

The objective of the trimming and packing procedure is that after each cell is packed, the remaining "packable" subdomain R' of R is always kept as a connected region. The rationale for maintaining the connectivity of R' is as follows. To pack spheres in a bounded domain R, two typical approaches have been used: (a) packing spheres layer by layer from the boundary of R toward its interior [41], and (b) packing spheres from the "center" of R, such as its medial axis, to the boundary [53–55,61]. Due to the shape irregularity of R, both approaches may fragment the "packable" subdomain R' into more and more disconnected regions; however, at the end of packing each such region, a small "unpackable" area may eventually remain that allows no further packing. It could fit more spheres if the "packable" subdomain R' is lumped together instead of being divided into fragments, which is what the trimming and packing procedure aims to achieve.

Due to the packing of its adjacent cells, the boundary of a cell $C(v)$ that is to be packed may consist of both line segments and arcs (of packed spheres). Hence it is needed to consider the problem of packing spheres in cells bounded by certain curves. In the trimming and packing phase, they presented several packing algorithms for different types of domain with or without a curved boundary. Their packing algorithms are based on certain lattice structures and allow the domain R to both translate and rotate. Their algorithms have fairly low time complexities. In certain cases, they even run in near linear time. Their algorithms can be easily generalized to congruent packing of other shapes, and are readily extended to higher dimensional spaces.

An interesting feature of the packings generated by the trimming and packing procedure is that the resulted spheres cluster together in the middle of the domain R, leaving some small unpackable areas scattered along the boundary of R. The "shake" procedure thus seeks to collect these small areas together by "blowing" the spheres to the boundary of R, in the hope of obtaining some "packable" region in the middle of R. They experimented quite a few techniques for efficiently shaking the packed spheres in R and compared their performances based on randomly generated input data.

Chen et al. [50] implemented their 2-D pack-and-shake algorithms and presented a set of experimental results. Their experiments showed that the packing algorithms consistently yield dense packings. Comparing with those optimization-based methods, their approaches improve dramatically on the running time with only a slight loss on the packing density.

The remainder of this chapter is organized as follows. In Section 78.2, we give some notations that will be used throughout this chapter. In Section 78.3, we present the algorithms in Ref. [50] for packing spheres in a single cell. Section 78.4 illustrates the ideas in Ref. [50] on how to partition the domain into cells. Section 78.5 discusses the extensions of the packing algorithms to higher dimensional space, and Section 78.6 discusses applications in treatment planning of radiosurgery. We present in Section 78.7 some experimental results from Ref. [50].

78.2 Preliminaries

Let $U = \{\vec{u_1}, \vec{u_2}, \ldots, \vec{u_d}\}$ be a set of d independent vectors in the d-D space E^d, and M be the $d \times d$ matrix formed by the d vectors. For any integer vector $\vec{x} = (x_1, x_2, \ldots, x_d)^T$ (i.e., each x_i is an integer), $M\vec{x}$ is a vector (or called *lattice point*) in E^d. The set of vectors so *generated* by using all integer vectors $\vec{x}$ forms a *lattice* L_U in E^d. M is called the *generator matrix* of L_U, and U is called the *basis*.

For each lattice L_U, there is a polyhedron B in E^d formed by the set of vertices, $v_0, v_1, \ldots, v_d, v_{d+1}$, where v_0 is the origin, $v_{d+1} = M \times (1, 1, \ldots, 1)^T$, and $v_i = M\vec{e_i}$ for $1 \leq i \leq d$ and $\vec{e_i} = (\underbrace{0, \ldots, 0}_{i-1}, 1, 0, \ldots, 0)^T$. B is called the *basic block* of L_U, which can be translated to form a tile of E^d. In 2-D, the basic block is a parallelogram, called the *basic parallelogram.*

We say that a lattice L_U admits a packing (in the whole space) of congruent spheres of radius r if the Euclidean distance between any pair of lattice points is no less than $2r$.

We denote a sphere packing of a bounded domain R as SP_R, and the densest sphere packing of R (i.e., a packing with the maximum number of spheres in R) as SP_R^{Max}. If the center of each sphere in SP_R is at a lattice point of L_U, then SP_R is called a lattice sphere packing of R, denoted as LSP_R. Similarly, we can define LSP_R^{Max}.

We denote a translation of an object O (e.g., a point, polygon and sphere) in E^d with offset vector $\Delta X = (\Delta x_1, \Delta x_2, \ldots, \Delta x_d)$ as $O + \Delta X$. ΔX is called the offset point.

78.3 Pack-and-Shake on a Polygonal Domain without Cell Partition

In this section, we introduce the algorithms in Ref. [50] for packing 2-D unit spheres in different types of simple polygon or polygonal domain R, under the assumption that the whole domain R is treated as a single cell (i.e., no further cell partitioning is done on R). The algorithms are based on certain lattice structures, and have low time complexities. Combining with a global *shake* refinement procedure, the resulted packings exhibit a fairly high density as shown by experimental results in a later section.

Let R be an n-vertex simple polygon or polygonal domain (i.e., a polygon with holes), and $U = \{\vec{u}_1, \vec{u}_2\}$ be the basis of a lattice L_U on the plane P such that L_U admits a unit sphere packing. Ideally, one would like to obtain the densest sphere packing SP_R^{Max}. However, as mentioned in Section 78.1, this problem is strongly NP-hard. Our goal thus is to seek efficient algorithms for producing dense packings SP_R.

Our approach is to first obtain the densest lattice unit sphere packing LSP_R^{Max}, and then use a shake procedure to globally adjust LSP_R^{Max} to generate a denser packing in R. Suppose that the plane P is already packed by infinitely many unit spheres, with each lattice point of L_U coincident with the center of a sphere. To obtain LSP_R^{Max} from the packing of P, we need to find a position and orientation of R on P such that R contains the maximum number of spheres from the packing of P. We discuss below two types of algorithms for computing the optimal position of R on P: translational algorithms that allow R to be translated, and rotational algorithms that allow R to be both translated and rotated.

78.3.1 2-D Lattice Packing with Translation

To produce efficiently the densest translational lattice sphere packing of R (denoted by $TLSP_R^{Max}$) in a given lattice L_U, we need to first identify a finite set S of spheres from the packing of P such that S contains at least one optimal solution for R. Once S is determined, we then seek an optimal position of R with respect to S. We solve this problem by reducing it to an interesting *thickest point problem*, and give an algorithm.

A frequently arising problem is to decide whether a sphere $s \in S$ is completely contained in R. To simplify this problem, we "shrink" the domain R by computing the Minkowski sum of a unit sphere s with the complement region $\overline{R}$ of R. The resulted region is denoted by R_-, that is, $R_- = \overline{\overline{R} \oplus s}$. Note that the boundary of R_- may consist of both line segments and arcs (each arc is associated with a reflex vertex of R). After the shrinking, a sphere s is contained in R if and only if the center c_s of s is inside R_-.

The shrunk domain R_- need not be topologically equivalent to the original domain R. When this is the case, R_- is broken into multiple disconnected components, each corresponding to a subdomain of R. We pack on each component independently of other components. Note that it is possible that the spheres in two different components interfere with each other. However, such interferences can be handled with some care. Hence, we assume in this section that the shrunk domain R_- is one connected component with n vertices. The next lemma shows that shrinking R can be done efficiently.

Lemma 78.1

An n-vertex polygonal domain can be shrunk in $O(n \log n)$ time.

Now we consider the problem of identifying the finite set S of spheres from the lattice L_U. Since the domain is already shrunk, we only need to identify a finite set of lattice points from L_U that gives rise to at least one optimal packing $TLSP_R^{Max}$. Let $v_1, v_2, \ldots, v_n$ be the n vertices of R_-. Let $R_- + (u, v)$ be a translation of R_- with offset point (u, v). Since the plane P can be tiled by translating infinite copies of the basic parallelogram of L_U, the following lemma holds.

Lemma 78.2

To obtain $TLSP_R^{Max}$, it is sufficient to translate R_- around inside the basic parallelogram.

Proof

Let $TLSP_R^{Max}$ be an optimal packing of R with a translation offset point $t_o = (u, v)$. Further, let BP_o be the basic parallelogram that contains the point t_o and $(\Delta x, \Delta y)$ be its offset from the basic parallelogram BP (i.e., $BP_o = BP + (\Delta x, \Delta y)$). Since each vertex u_i of BP_o, $i = 0, 1, 2, 3$, is a lattice point, $\vec{u_i} = M\vec{x_i}$ for some integer vector $\vec{x_i}$. In particular, $(\Delta x, \Delta y)^T = M\vec{x_0}$. We map t_o to a point $t'_o = (u - \Delta x, v - \Delta y)$. Clearly, t'_o is inside BP.

Below we prove the claim that for every sphere s in $TLSP_R^{Max}$ centered at $s_c = (x_{s_c}, y_{s_c})$, there is another sphere s' centered at $s'_c = (x_{s'_c}, y_{s'_c}) = s_c - (\Delta x, \Delta y)$, such that s' is contained in $R_- + t'_o$ (i.e., the translation of R_- with an offset point t'_o).

To show the existence of such a sphere s' in L_U, we observe that s_c is a lattice point. Hence $\vec{s_c} = M\vec{x_{s_c}}$ for some integer vector $\vec{x_{s_c}}$. Since $s'_c = s_c - (\Delta x, \Delta y)$, $\vec{s'_c} = \vec{s_c} - (\Delta x, \Delta y)^T = M\vec{x_{s_c}} - M\vec{x_0} = M(\vec{x_{s_c}} - \vec{x_c})$. Thus, both $\vec{x_{s_c}}$ and $\vec{x_0}$ are integer vectors, and $\vec{x_{s_c}} - \vec{x_0}$ is also an integer vector. Therefore, s'_c is a lattice point of L_U.

To prove that s' is contained in $R_- + t'_o$, we show that the distance from s_c to any vertex v_i of $R_- + (u, v)$ is the same as that from s'_c to v_i in $R_- + t'_o$. The coordinates of v_i is $v_i + (u, v)$ in $R_- + (u, v)$ and $v_i + t'_o = v_i + (u, v) - (\Delta x, \Delta y)$ in $R_- + t'_o$. The vector $\overrightarrow{v_i s_c}$ from v_i (in $R_- + (u, v)$) to s_c is $\vec{s_c} - (\vec{v_i} + (u, v)^T)$, and the vector $\overrightarrow{v_i s'_c}$ from v_i (in $R_- + t'_o$) to s'_c is $\vec{s_c} - (\Delta x, \Delta y)^T - (\vec{v_i} + (u, v)^T - (\Delta x, \Delta y)^T) = \vec{s_c} - (\vec{v_i} + (u, v)^T)$. Thus the two distances are the same, and the claim is true. □

The above lemma suggests that to find $TLSP_R^{Max}$, it is sufficient to consider the set of spheres (more accurately, the set of lattice points), which are away from R_- only by a basic parallelogram. As we move the offset point $f = (u, v)$ for R_- inside the basic parallelogram BP, lattice points may go in and out of R_-, causing the number of spheres contained in R to change. Thus, we need to identify the set S of lattice points that may cross the boundary of R_- while f is moving around BP.

To compute S, we consider a segment $\overline{ab}$ of R_- (an arc can be handled similarly). The lattice points, which may cross or touch $\overline{ab}$ can be determined as follows. Let (x_a, y_a) and (x_b, y_b) be the coordinates of points a and b, respectively. We first determine to which parallelogram, BP_a, a belongs. (This can be easily done in $O(1)$ time by decomposing $\vec{a}$ into two vectors $\vec{a_1}$ and $\vec{a_2}$ along the directions of the basis vectors $\vec{u_1}$ and $\vec{u_2}$ and dividing $\vec{a_i}$, $i = 1, 2$, by $\vec{u_i}$ to obtain the integer vector $\vec{x_0} = (i_0, j_0)^T$.) Then we check the four lattice points of BP_a to see whether they cross $\overline{ab}$ while moving f in BP, and add each crossing point into a set $S_{\overline{ab}}$. For each lattice point $s_i = M\vec{x_i}$ in $S_{\overline{ab}}$, we further check the following three lattice points: $M(\vec{x_i} + (1, 0)^T)$, $M(\vec{x_i} + (0, 1)^T)$, and $M(\vec{x_i} + (1, 1)^T)$, and add the crossing points into $S_{\overline{ab}}$. By repeating this procedure, all lattice points which may cross $\overline{ab}$ are put into $S_{\overline{ab}}$.

Repeating the above procedure for each segment and arc of R_-, we obtain a collection of point sets. The union of them forms the sought lattice point set S. The size of S is roughly $O(m)$, where m is the total number of lattice points along the boundary of R_- in $TLSP_R^{Max}$.

Once S is obtained, we need to determine an optimal translational position for R_-. That is, we need to determine a point f_o in BP for the offset point f such that $R_- + f_o$ contains the maximum number of points in S. For this purpose, we compute, for each lattice point $s \in S$, a region s_f inside BP such that when f moves inside s_f, s is contained in $R_- + f$. The region s_f is called the *containing region* of s. The optimal offset point f_o is thus a point that is covered by the maximum number of containing regions. Below we discuss how to compute the containing region of each lattice point of S.

Assume that $\vec{s} = M\vec{x_s}$ is a member of $S_{\overline{ab}}$. (The case in which $\overline{ab}$ is an arc can be handled similarly.) We first compute the locus of the offset point f in BP when the boundary of $R_- + f$ touches s. Each point w on $\overline{ab} + f$ can be represented by its corresponding vector $\vec{w} = \vec{a} + \vec{f} + t(\vec{b} - \vec{a})$ for some $0 \leq t \leq 1$. To make $\overline{ab} + f$ touch s, we must have some point $\vec{w} = \vec{s}$. This means that there is a value $t \in [0, 1]$ such that $\vec{w} = \vec{a} + \vec{f} + t(\vec{b} - \vec{a}) = \vec{s}$. Thus, the locus of f can be determined by the equality $\vec{f} = \vec{s} - \vec{a} + t(\vec{a} - \vec{b})$, $t \in [0,1]$. (Note that the slope of the locus is the same as that of $\overline{ab}$.) The locus of f is trimmed by the boundary of BP if it goes outside of BP. For each segment $\overline{uv}$ whose $S_{\overline{uv}}$ contains s, we compute the locus of f. All these loci, possibly together with the boundary of BP, form the containing region s_f of s.

The containing region s_f has some interesting properties.

Property 78.1

If R_- is convex, then all containing regions are convex. If R_- contains a convex hole, then the containing region of a lattice point generated by an edge of this hole is the complement of a convex region in BP.

The number of edges in each containing region is less than $n + 4$. If all edges of R_- are "long" (i.e., $\geq |\vec{u}_1 + \vec{u}_2|$), then each lattice point can touch or cross only a constant number of edges of R_-, and thus each containing region has a constant number of edges.

The above approach for generating containing regions can be extended to the case in which R_- is bounded by a set of algebraic curves.

Property 78.2

If R_- is bounded by a set of algebraic curves, then each containing region is also bounded by a set of algebraic curves. Further, each curve of R_- has the same degree as its corresponding curve of the containing region.

Since the shrinking procedure does not change the degrees of the curves of R_-, based on Property 78.2, we can now consider the translational lattice packing problem in a domain R bounded by algebraic curves. Once the set of containing regions is generated, the packing problem is reduced to the following *thickest point problem*, where the *thickness* of a point is defined as the number of containing regions covering it.

Problem 78.1

Given a set F of connected regions on the plane, with each region bounded by a set of algebraic curves, find the thickest point on the plane.

To solve this problem, let Γ be the set of algebraic curves bounding the regions in F, and $N = |\Gamma|$. Obviously, Γ forms an arrangement A on the plane. If the degree of each curve in Γ is bounded by a constant, then any two such curves intersect only a constant number of times. Thus, we can use well-known arrangement algorithms [61,62] to first construct A in $O(N \log N + K)$ time, where K is the total number of intersections in A. In the worst case, $K = O(N^2)$. The optimal point can then be found by traversing the cells of A and computing the thickness of each cell.

Lemma 78.3

If every edge in Γ is an algebraic curve with constant degree, then the thickest point problem on F can be solved in $O(N \log N + K)$ time.

Now, we go back to the sphere packing problem. As we have shown, a containing region is computed from R_- and a lattice point in S. If R is bounded by a set of degree-bounded algebraic curves, we have the following lemma.

Lemma 78.4

Given a domain R with its boundary edges being algebraic curves of constant degrees, the $TLSP_R^{Max}$ problem can be solved in $O(N \log N + K)$ time, where N is the number of edges of the generated containing regions, and K is the size of the arrangement formed by the containing regions.

In the above lemma, $N = n \times m$ in the worst case, where n is the size of R and m is the total number of spheres along the boundary of R in $TLSP_R^{Max}$. Note that, in practice, N may be much smaller than $n \times m$. For example, for the cases in which all bounding edges of R are "long" edges (i.e., $\geq |\vec{u}_1 + \vec{u}_2|$), or "short" edges are separated by long edges, the containing region of each lattice point has only a constant number of edges, and thus N is roughly $O(n + m)$.

78.3.2 2-D Lattice Packing with Translation and Rotation

For a domain R, the translational lattice packing may not yield the densest lattice packing due to the given orientation of R. Thus, we like to study the rotational lattice packing problem that allows R to be translated and rotated. We denote the densest rotational lattice sphere packing of R by $RLSP_R^{Max}$.

To compute $RLSP_R^{Max}$, similar to the translational lattice packing, we first shrink R to obtain R_-. We then identify a set S of lattice points, which may cross the boundary of R_- while translating and rotating R_-. Finally, we determine the optimal position and orientation of R.

To compute S, we consider all possible lattice points that may cross an edge of R_- while translating and rotating R_-. The rotation of R_- may cause a large set of lattice points to cross the boundary of R_-. For each boundary edge $\overline{ab}$ of R_-, the number of crossing lattice points of $\overline{ab}$ is roughly proportional to the area "swept" by $\overline{ab}$ during the rotation, which clearly depends on the location of the rotation center. Thus, the problem of minimizing the set S is reduced to finding an optimal rotation center that minimizes the total area swept by all $O(n)$ edges of R_-. We call this the *annuli minimization problem.*

To find an optimal rotation center, we consider an edge $\overline{ab}$ of R_-. Suppose that the rotation center is $r_o = (x, y)$. Then the rotation of $\overline{ab}$ sweeps the plane and forms an annulus $nu_{\overline{ab}}$. The area of $nu_{\overline{ab}}$ is equal to $\pi(r_{out}^2 - r_{in}^2)$, where r_{out} and r_{in} are the radii of its inner and outer circles. Precisely, r_{out} is the distance from r_o to the furthest point of a and b, and r_{in} is the distance from r_o either to the supporting line l_{ab} of $\overline{ab}$ or to the nearest of a and b. There are four different cases. To distinguish these cases, we draw three lines: one is the bisector bs_{ab} of $\overline{ab}$, and the other two, l_a and l_b, are parallel to bs_{ab} and passing through a and b respectively. In each of the four regions partitioned by these three lines, the area of the annulus is either a linear or quadratic function of x and y. Thus, the optimal rotation center can be computed by first constructing an arrangement A' from the $O(n)$ partitioning lines obtained from all edges of R_-, and then in each convex cell of A', finding an optimal rotation center point by solving the associated quadratic minimization problem on that cell. The objective function for each cell of A' can be updated in an online fashion.

Lemma 78.5

For a domain R_- of n edges, the annuli minimization problem can be solved by reducing it to solving $O(n^2)$ quadratic minimization problems. Furthermore, these $O(n^2)$ optimization problems (including both the objective functions and constraints) can be generated in altogether $O(n^2)$ time.

Note that the above approach for computing the rotation center is also applicable to the case when the rotation angle is $< 2\pi$ (e.g., for a hexagonal lattice, it is sufficient to rotate R_- between 0 and $\frac{\pi}{3}$).

Once the rotation center is located, we translate the origin of the coordinate system to this center. (This changes the coordinates of the vertices of R_-.) The set S of crossing lattice points, can then be computed from those lattice points, which are either contained by any of the n annuli (or a portion of an annulus if the rotation is $< 2\pi$) or within a distance of a basic parallelogram away from the lattice points contained by the annuli (for translation). The total time for generating S is $O(|S|)$. Similar to our translational lattice packing approach, the rotational lattice packing algorithm needs to identify the containing regions for all lattice points in S with respect to R_-. With the motions of both translation and rotation, the space for the containing regions in this case becomes 3-D (the third dimension, called the α-axis, represents the rotation angles). To generate the set of 3-D containing regions, we first compute the set of the 2-D initial containing regions (i.e., with a rotation angle $\alpha = 0$). Due to the rotation, each edge of a 2-D containing region becomes a 3-D *spiral* surface patch. To see that, we consider a segment $\overline{ab}$ of R_- and a lattice point $s \in S$. Each point p of $\overline{ab}$ can be represented as $\vec{p} = \vec{a} + \vec{f} + tW(\vec{b} - \vec{a})$, $0 \leq t \leq 1$, where f is the offset point, W is the rotation matrix with

$$W = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

and θ is the rotation angle. The locus of the 3-D point (f, θ) forms a surface SF_{ab} when $\overline{ab}$ is translated and rotated as it remains touching s. The surface SF_{ab} can be expressed by the equality $\vec{f} = \vec{s} - \vec{a} + tW(\vec{a} - \vec{b})$, $0 \leq t \leq 1$.

In 3-D, the basic parallelogram BP becomes a polygonal cylinder C. Surface patches, which go outside of C are truncated at the boundary of C. Denote the set of (truncated) surface patches and the boundary of C as Γ. The following lemma shows some nice property of the surface patches in Γ.

Lemma 78.6

The surface patches in Γ are all pseudoplanes. That is, any two surface patches in Γ intersect at a continuous curve (if they intersect each other), and any three surface patches intersect at no more than one point.

With the above lemma, we solve the thickest point problem in 3-D by using a space-sweeping algorithm along the α-axis. Note that each containing region in 3-D is a polygonal cylinder spiraling up along the α-axis. The thickest point is covered by the maximum number of such cylinders. The space-sweeping algorithm first constructs the planar arrangement A_0 corresponding to the rotation angle $\alpha = 0$, and computes the thickness map for each cell of A_0. To determine the next event point, we consider an edge $e_i \in A_0$. Let e_{i-1} and e_{i+1} be the neighboring edges of e_i along a cell of A_0. Denote the three surface patches generated by the rotation of the three edges as SF_i, SF_{i-1}, and SF_{i+1}. We compute the intersection point $v_{i,i-1}$ (resp., $v_{i,i+1}$) between the bounding curves of SF_i and SF_{i-1} (resp., SF_{i+1}) with the other surface patch, and the intersection point v of the three surface patches. All these intersection points are inserted into a priority queue based on their α-coordinates. The next event point is the intersection point in the priority queue with the smallest α-coordinate. At each event point, the algorithm may generate new event points, and it identifies whether a new cell is created or an old cell disappears. For these two cell changing cases, the algorithm needs to update the thickness map. The thickness of a new cell can be computed from its neighboring cells in $O(1)$ time. In this way, the sweeping algorithm spends only $O(1)$ time to compute each vertex of the whole 3-D arrangement A of Γ, and the total computation cost can be charged to the vertices of A with a logarithmic cost per vertex. Thus, we have the following lemma.

Lemma 78.7

The space-sweeping algorithm finds the thickest point in 3-D in $O((K + N) \log N)$ time, where K is the size of the 3-D arrangement A and $N = |\Gamma|$.

In the worst case, $K = O(N^3)$. Based on our experiments, K is normally much smaller than $O(N^3)$.

Remark

The above approach for finding rotational lattice sphere packing can be extended to the case in which R is bounded by a set of algebraic curves with constant degrees.

78.3.3 Shaking a Lattice Packing

The lattice packings produced in Sections 78.3.1 and 78.3.2 have an interesting property. That is, the resulted spheres cluster in the middle of the domain R. For certain lattice structures, for example, the 2-D hexagonal lattice whose basis is $\{(2, 0)^T, (1, \sqrt{3})^T\}$, the packings are very tight in the middle. Locally, the packing may even be optimal in the middle. But globally, there may still be some small "unpackable" areas scattered along the boundary of R. To make use of these small areas, we apply a "shake" procedure, which tries to "blow" the spheres from the "center" of R to its boundary and gather these unused small areas to the center for packing more spheres.

From experiments, we observed that unused yet unpackable areas frequently occur around the corners (vertices) of the domain. Thus, we first push a sphere to each unoccupied corner. Next, we repeatedly apply two procedures, "move" and "drop". The two procedures could be viewed as making the movements of spheres in a special field of forces. The moving directions of the spheres could be different from sphere to sphere. Unlike some physical simulation such as in Ref. [33], in the shake procedure, a sphere could "pass through" some baffled spheres to reach a stable point under the effect of the field.

Several different shake procedures are implemented and tested in Ref. [50], based on different ways of choosing the moving directions and different criteria for the movements. For example, one technique is to scatter from an "anchor point," which is derived from the positions of the spheres. Intuitively, the "anchor point" repulses a sphere while the nearest edge attracts it. Each force is associated with a weight, which helps determine the sphere movement. Different shake procedures can be used one after another to yield denser packing.

We continue the shaking until no more sphere can be added. During the shaking process, whenever we detect that an empty space large enough for a sphere shows up, we put a new sphere to take that space. Experiments show that the shake procedures work very well. It will improve the packing density significantly, especially for those packings with low density. Furthermore, shaking can be done efficiently (in practice, it often takes several or tens of seconds) by moving spheres layer by layer.

78.4 Trimming and Packing

In the previous section, we use the domain R as a whole to pack spheres. The packings so generated are very dense for "fat" or convex domains, since the initial lattice packings already occupy most area in the middle of such domains and leave only very small unused areas along their boundaries.

However, this strategy does not always work well, especially for irregular-shaped domains. Due to its local structures, different parts of a domain R may require different translational and rotational lattice packings. Treating R as a single cell may make the overall lattice packing density very low. In Figure 78.1(a), for example, the domain R consists of multiple "strips" connected together, and each strip can pack a row of spheres; it is possible that no matter which translation or rotation lattice packing is used, the packing density remains very low if the whole R is treated as one cell. A better approach is to partition R into multiple "nice" cells, and pack each cell somewhat independently. Figure 78.1(b) shows an optimal packing based on a partition of R.

Another reason for partitioning a domain into multiple cells is to achieve a better time efficiency. In Section 78.3, the packing algorithms all take superlinear time in terms of the number N of edges or surfaces of the containing regions. N is between $O(n + m)$ and $O(nm)$, where n is the number of edges of R_{-} and m is the number of involved crossing lattice points. Partitioning R into smaller "fat" and convex cells makes the values of n and m for packing each cell much smaller than those for R, and thus speeds up the overall running time significantly.

To achieve a faster and denser packing, we use a procedure called *trimming and packing*. This procedure first partitions the domain R into a set of triangles or trapezoids. Then the neighboring triangles or trapezoids are merged to form larger convex cells. The set of cells define a dual graph G_D (each vertex v of G_D is for a cell $C(v)$, and an edge connects two vertices if their d-D cells share a $(d - 1)$-D face). To pack spheres in all cells, the procedure repeatedly removes the lowest degree vertex v from G_D and packs $C(v)$ by using the algorithms in Section 78.3.

One problem that needs to be solved in the trimming and packing procedure is how to join the packings of two adjacent cells, say $C(v)$ and $C(u)$. Note that independently packing $C(v)$ and $C(u)$ may leave some small room along the edge separating the two cells. If there are many cells, then the total unused area could

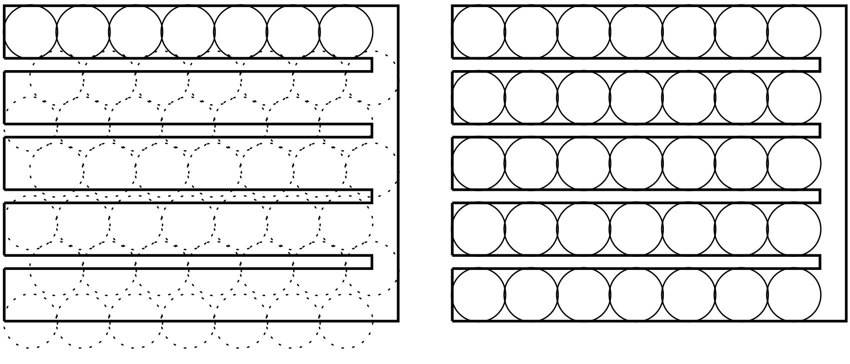

(a) (b)

FIGURE 78.1 (a) Lattice packing on R as a whole may not yield a dense packing. (b) An optimal packing obtained by using lattice packing in two subdomains.

be significant. To remedy this problem, when packing a cell, say $C(u)$, we actually pack the union of $C(u)$ and the small areas left by its neighbors that have already been packed. The boundary of each packed cell is a curve formed by the outer boundary of the packed spheres. This means that the actual packable region for $C(u)$ is bounded by a mix of line segments and circular arcs.

One way to solve this problem is to use the algorithms for handling domains bounded by algebraic curves. This approach normally generates dense packing, but takes much longer time than packing a polygonal domain. From a practical point of view, a faster algorithm is as follows. First of all, R is partitioned into cells with a small number of boundary edges. Then the efficient translation algorithm is used to obtain a translational lattice packing for each cell. Finally, a shake procedure is used on every pair of neighboring cells.

Remark

To reduce the running time of the trimming and packing procedure, one possible way is to compute a convex decomposition that minimizes the total length of the diagonal edges, since such a partition minimizes the number of circular arcs and crossing lattice points. There are several sophisticated convex decomposition algorithms (e.g., Keil and Snoeyink's algorithm [63]). But, all such algorithms have quite large time bounds, probably not easy to implement, and may not work for domains with holes. Hence, we use a simple algorithm for convex decomposition.

An important feature of our trimming and packing procedure is that the remaining "packable" subdomain is always a connected component. This prevents the algorithm from producing many "unpackable" small areas, a problem occurring in some commonly used approaches [41,53–55].

78.5 Extensions to Three or Higher Dimensions

The sphere packing algorithms in previous sections can be extended in several directions, which we sketch in this section. These include extending the lattice packing algorithms to 3-D and high dimensions, and applying the sphere packing algorithms to treatment planning of radiosurgery.

To pack spheres in a 3-D domain, we also use the trimming and packing procedure to partition the domain into convex cells. For each cell, we use lattice packing algorithms to find a good initial packing, and shaking procedures to improve the packing. Since the trimming and packing, shaking, and shrinking algorithms are similar to those for 2-D, we focus on the lattice packing step. Further, our experiments suggest that for convex domains, the quality of translation packings is very close to that of rotation packings. Thus, we only discuss the 3-D translational lattice packing in a convex cell.

Suppose after shrinking, the convex domain R consists of n vertices. Further, assume that the boundary of R_- is triangulated. Since a lemma similar to Lemma 78.2 holds in 3-D (which can be proved similarly), the set S of crossing lattice points can be easily computed. Thus, to compute the containing regions, we only need to consider a triangle Δ_{abc} on the boundary of R_- against a lattice point $s \in S$.

Let w be any point of $\Delta_{abc} + f$, where f is the offset point of R_- inside the basic block of L_U. Then w can be expressed as $\vec{w} = \vec{a} + \vec{f} + t_1(\vec{b} - \vec{a}) + t_2(\vec{c} - \vec{a})$ with $t_1 \in [0,1]$, $t_2 \in [0,1]$, and $t_1 + t_2 \in [0, 1]$. The locus of f, when Δ_{abc} touches s, is determined by $\vec{w} = \vec{s}$, and can be expressed by $\vec{f} = \vec{s} - \vec{a} + t_1(\vec{a} - \vec{b}) + t_2(\vec{a} - \vec{c})$. Clearly, this is a triangle with the same normal as Δ_{abc}.

The optimal 3-D translational lattice packing $TLSP_R^{Max}$ can be obtained by finding the thickest point inside the basic block. Using a space-sweeping algorithm, we have the following lemma.

Lemma 78.8

For a 3-D convex polyhedral domain with n vertices, the thickest point problem can be solved in $O((K + N)\log N)$ time, where N is the number of faces of the set of containing regions and K is the size of the 3-D arrangement generated by the set of N faces.

It is also possible to extend the above approach for translational lattice sphere packing to higher dimensional spaces.

78.6 Applications in Gamma Knife Treatment Planning

The congruent sphere packing algorithms can be used as a key procedure for solving the sphere packing problem arising in Gamma Knife surgical treatment planning. Based on a common approach used in practice, we can first put congruent balls of the largest available size into the domain, and then repeat for the next size balls, until no more balls can be put into the domain. Note that our algorithms for domains bounded by algebraic curves are especially useful in this setting (since packing larger balls leaves subdomains bounded by arcs). This common approach is intended for reducing the total number of balls used and thus shortening the treatment time.

After the balls are placed into the domain, another problem in Gamma Knife surgical treatment planning is to determine how much radiation (i.e., the weight) each ball should deliver. Note that the radiation in a ball also affects the surrounding uncovered parts of the tumor domain. To completely destroy a tumor, every point p of the tumor domain should receive a prescribed amount of radiation. In particular, if p is not in a ball, then it must receive the required radiation from the "cross-firing" of radiation by nearby balls. This is a weight assignment problem for balls after a packing is done.

To solve the weight assignment problem of balls, we use the following approaches for 2-D domains (3 or higher dimensional domains can be handled similarly). We can first approximate the function of radiation spread from a sphere s by a set of circles whose centers are all at the center of s and assume that all points between any two such consecutive circles receive the "same" amount of radiation from s, the set of circles, defined by all packed spheres, forms a 2-D arrangement A. The amount of radiation in each cell of A is a linear sum determined by the weights of the spheres. Since dose distribution is known in advance, we can determine the expected radiation for each cell of A. The weight of each sphere can thus be determined by solving a linear programming LP. In this LP, each cell of A contributes a linear constraint indicating that the amount of radiation for this cell should not excess (or be too much lower than) the expected radiation. The objective of the LP is to minimize the total difference between the expected and actually received radiation over all cells of A.

78.7 Experimental Studies

In this section, we show some experimental results on the 2-D pack-and-shake sphere packing algorithms from Ref. [50]. All implementations use the hexagonal lattice as an example. The implementation is on Sun Ultra Sparc 30 workstations using the C++ based library LEDA 4.1. Experiments suggest that the aforementioned algorithms produce reasonably dense packings for polygonal domains of almost all shapes.

Four algorithms were tested: *Translation*, *Translation and Shake* (*TS*), *Rotation*, and *Rotation and Shake*. The first two algorithms ran very fast, normally in about 1 min. However, the last two algorithms took about 15 h. An explanation for such long execution time is that these two algorithms need to compute the 3-D arrangement of $O(N)$ complicated surfaces as discussed in Section 78.3. Furthermore, computing such 3-D arrangements involves solving many nonlinear equation systems, which takes nontrivial numerical manipulations. The data were obtained by taking the average of many examples. The numbers of spheres used in those packing examples range from 20 to 300.

The results show that in most cases, *Rotation and Shake* gave a better packing quality, but hours to terminate. While *TS* sometimes gave a little worse quality than *Rotation and Shake*, it ran much faster. The packing quality differed from 1.2 to 4.65%. Thus, there is a tradeoff between the execution time and packing quality.

Comparisons were also done between the above algorithms and some known results obtained by using optimization methods [60] on domains of triangles and squares. In particular, we packed spheres in the unit square by using the algorithm TS, and compared the results with those in Ref. [60]. Note that the problem studied in Ref. [60] is somewhat different: Given a domain and an integer k, pack k congruent spheres in the domain to maximize their diameter. Some nonconvex programming methods are used in Ref. [60] for that problem, and their algorithms generally run in hours. We used the output diameters of

TABLE 78.1 Comparison of Packing Results on Square Domains

Diameter	Spheres by TS	Spheres by Ref. [60]
0.166454626	46	50
0.174459361	43	46
0.178639224	43	44
0.188175077	39	40
0.196238101	35	37
0.205021908	31	34
0.213082353	30	32
0.217547292	30	31
0.226882901	28	29
0.235849528	25	27
0.254333095	23	24
0.267958402	20	22

Source: From Chen, D. Z. et al., Algorithms for congruent sphere packing and applications, *Proc. 17th Annual ACM Symp. on Computational Geometry*, 2001, pp. 212–221.

Ref. [60] as our input diameters. This comparison thus may not be completely fair, but we believe it is still meaningful. Table 78.1 shows the different number of spheres packed by our TS algorithm and the algorithms in Ref. [60] in the unit square with different diameters. It shows that our packing qualities are no more than 10% worse than those in Ref. [60]. But our TS packing algorithm only ran in less than 5 s (compared to the hours of Ref. [60]).

References

[1] Bär G. and Iturriaga, C., Rectangle packing in polynomial time, *Proc. 5th Canad. Conf. Comput. Geom.*, 1993, p. 455.

[2] Coffman, E. G. Jr., and Lagarias, J. C., Algorithms for packing squares: A probabilistic analysis, *SIAM J. Comput.*, 18(1), 166, 1989.

[3] Conway, J. H. and Sloane, N. J. A., *Sphere Packings, Lattices and Groups*, Springer, New York, 1988.

[4] Fekete, S. P. and Schepers, J., On More-Dimensional Packing I: Modeling, Technical report, ZPR 97-288, Website http://www.zpr.uni-koeln.de/∼paper.

[5] Fekete, S. P. and J. Schepers, J., On More-Dimensional Packing II: Bounds, Technical report, ZPR 97-289, Website http://www.zpr.uni-koeln.de/∼paper.

[6] Fekete, S. P. and Schepers, J., On More-Dimensional Packing III: Exact Algorithm, Technical report, ZPR 97-290, Website http://www.zpr.uni-koeln.de/∼paper.

[7] Formann, M. and Wagner, F., A packing problem with applications to lettering of maps, *Proc. ACM Symp. Comput. Geom.*, 1991, p. 281.

[8] Füredi, Z., The densest packing of equal circles into a parallel strip, *Disc. Comput. Geom.*, 6, 95, 1991.

[9] Graham, R. L. and Sloane, N. J. A., Penny-packing and two-dimensional codes, *Disc. Comput. Geom.*, 5, 1, 1990.

[10] Hilbert, D., *Mathematische Probleme*, Archiv. Math. Phys. 1, 1901, 44–63 and 213–237 (Gesamm. Abh., III, 290-329. English translation in BAMS 8 (1902), 437–479 *Proc. Sympo. in Pure Math.*, 28 (1976), 1–34.

[11] Kabatiansky, G. A. and Levenshtein, V. I., Bounds for packings on a sphere and in space, *Problemy Peredachi Informatsii*, 14(1), 3, 1978.

[12] Kyperberg, G. and Kuperberg, W., Double-lattice packings of convex bodies in the plane, *Disc. Comput. Geom.*, 5, 389, 1990.

[13] Leech, J., Sphere packing and error-correcting codes, *Canadian J. Math.*, 23, 718, 1971.

[14] Maranas, C. D., Floudas, C. A., and Pardalos, P. M., New results in the packing of equal circles in a square, *Disc. Math.*, 142, 287, 1995.

[15] Mckenna, M., O'Rourke, J., and Suri, S., Finding maximal rectangles inscribed in an orthogonal polygon, *Proc. Allerton Conf. Commun. Control Comput.*, 1985, p. 486.

[16] Megiddo, N. and Supowit, K. J., On the complexity of some common geometric location problems, *SIAM J. Comput.*, 13(1), 182, 1984.

[17] Minkowski, H., Diskontinuitätsbereich für arithmetische Aequivalenz, *J. Reine Angew. Math.* 129, 220, 1905.

[18] Rogers, C. A., *Packing and Covering*, Cambridge University Press, Cambridge, 1964.

[19] Tóth, G. F. and Kuperberg, W., A survey of recent results in the theory of packing and covering, *New Trends in Disc. and Computational Geom.*, 10, Springer, Berlin, 1993, p. 251.

[20] Baker, B. S., Brown, D. J., and Katseff, H. K., A 5/4 algorithm for two-dimensional packing, *J. Algorithms*, 2, 348, 1981.

[21] Baker, B. S., Coffman, E. G., Jr., and Rivest, R. L., Orthogonal packing in two dimensions, *SIAM J. Comput.*, 9(4), 846, 1980.

[22] Baker, B. S. and Schwarz, J. S., Shelf algorithms for two-dimensional packing problems, *SIAM J. Comput.*, 12(3), 508, 1983.

[23] Coffman, E. G., Jr., Garey, M. R., Johnson, D. S., and Tarjan, R. E., Performance bounds for level-oriented two-dimensional packing algorithms, *SIAM J. Comput.*, 9, 808, 1980.

[24] Coffman, E. G., Jr., and Shor, P. W., Average-case analysis of cutting and packing in two dimensions, *Eur. J. Oper. Res.*, 44, 134, 1990.

[25] Daniels, K. M. and Milenkovic, V. J., Column-based strip packing using ordered and compliant containment, *Proc. ACM Workshop on Applied Computational Geom.*, 1996, p. 33.

[26] Daniels, K. M. and Milenkovic, V. J., Multiple translational containment, part I: an approximation algorithm, *Algorithmica*, 19, 148, 1997.

[27] Daniels, K. M., Milenkovic, V. J., and Roth, D., Finding the maximum area axis-parallel rectangle in a simple polygon, *Comput. Geom.: Theor. Appl.*, 7, 125, 1997.

[28] Dyckhoff, H., A topology of cutting and packing problems, *Eur. J. Oper. Res.*, 44, 145, 1990,

[29] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.

[30] Golan, I., Performance bounds for orthogonal oriented two-dimensional packing algorithms, *SIAM J. Comput.*, 10(3), 571, 1981.

[31] Heistermann, J. and Lengauer, T., Efficient automatic part nesting on irregular and inhomogeneous surfaces, *Proc. SODA,* 1993, p. 251.

[32] Li, Z. and Milenkovic, V. J., Compaction and separation algorithms for nonconvex polygons and their applications, *Eur. J. Oper. Res.*, 84, 539, 1995.

[33] Milenkovic, V. J., Position-based physics: animating and packing spheres inside polyhedra, *Proc. Canad. Conf. Comput. Geom.*, 1995, p. 79.

[34] Milenkovic, V. J., Translational polygon containment and minimal enclosure using linear programming based restriction, *Proc. STOC*, 1996, p. 109.

[35] Milenkovic, V. J., Multiple translational containment, part II: exact algorithms, *Algorithmica*, 19, 183, 1997.

[36] Milenkovic, V. J., Densest translational lattice packing of non-convex polygons, *Proc. Symp. on Computational Geom.*, 2000, p. 280.

[37] Steinberg, A., A strip-packing algorithm with absolute performance bound 2, *SIAM J. Comput.*, 26(2), 401, 1997.

[38] Bern, M., Mitchell, S., and Ruppert, J., Linear-size nonobtuse triangulation of polygons, *Proc. ACM Symp. on Comput. Geom.*, 1994, p. 221.

[39] Cheng, S.-W., Dey, T. K., Edelsbrunner, H., Facello, M. A., and Teng, S.-H., Silver exudation, *Proc. ACM Symp. on Comput. Geom.*, 1999, p. 1.

[40] Chew, L. P., Guaranteed-quality Delaunay meshing in 3D (short version), *Proc. ACM Symp. on Comput. Geom.*, 1997, p. 391.

[41] Li, X.-Y., Teng, S.-H., and Üngör, A., Biting: advancing front meets sphere packing, *Int. J. Numerical Meth. in Eng.*, 49(1-2), 61–81, 2000.

[42] Miller, G. L., Talmor, D., Teng, S.-H., and Walkington, N., A Delaunay based numerical method for three dimensions: generation, formulation and partition, *Proc. STOC*, 1995, p. 683.

[43] Ruppert, J., A new and simple algorithm for quality 2-dimensional mesh generation, *Proc. SODA*, 1992, p. 83.

[44] Shewchuk, J. R., Tetrahedral mesh generation by Delaunay refinement, *Proc. ACM Symp. on Comput. Geom.*, 1998, p. 86.

[45] Hochbaum, D. S. and Maass, W., Approximation schemes for covering and packing problems in image processing and VLSI, *JACM*, 32(1), 130, 1985.

[46] Karp, R. M., Luby, M., and Marchetti-Spaccamela, A., A probabilistic analysis of multidimensional bin-packing problems, *Proc. STOC*, 1984, p. 289.

[47] Li, K. and Cheng, K. H., On three-dimensional packing, *SIAM J. Comput.*, 19(5), 847, 1990.

[48] Baur, C. and Fekete, S. P., Approximation of geometric dispersion problems, *Proc. APPROX*, 1998, p. 63.

[49] Fowler, R. J., Paterson, M. S., and Tanimoto, S. L., Optimal packing and covering in the plane are NP-complete, *Inf. Proc. Lett.*, 12(3), 133, 1981.

[50] Chen, D. Z., Hu, X., Huang, Y., Li, Y., and Xu, J., Algorithms for Congruent Sphere Packing and Applications, *Proc. ACM Symp. on Comput. Geom.*, 2001, p. 212.

[51] Shepard, D. M., Ferris, M. C., Ove, R., and Ma, L., Inverse treatment planning for gamma knife radiosugery, *Medical Physics*, 27(12), 2748, 2000.

[52] Sutou, A. and Dai, Y., Global optimization approach to unequal sphere packing problems in 3D, *J. Opt. Theor. Appl.*, 114(3), 671, 2002.

[53] Bourland, J. D. and Wu, Q. R., Use of shape for automated, optimized 3D radiosurgical treatment planning, *SPIE Proc. Int. Symp. on Medical Imaging*, 1996, p. 553.

[54] Wu, Q. R., Treatment Planning Optimization for Gamma Unit Radiosurgery, Ph.D. thesis, The Mayo Graduate School, 1996.

[55] Wang, J., Packing of unequal spheres and automated radiosurgical treatment planning, *J. Comb. Opt.*, 3, 453, 1999.

[56] Mount, D. M. and Silverman, R., Packing and covering the plane with translates of a convex polygon, *J. Algorithms*, 11, 564, 1990.

[57] Graham, R. L. and Lubachevsky, B. D., Dense packing of equal disks in an equilateral triangle: from 22 to 34 and beyond, *Electron. J. Comb.*, 2, #A1, 1995.

[58] Graham, R. L. and Lubachevsky, B. D., Repeated patterns of dense packings of equal disks in a square, *Electron. J. Comb.*, 3, #R16, 1996.

[59] Friedman, E., Packing unit squares in squares: a survey and new results, *Electron. J. Comb.*, 5, #DS7, 1998.

[60] Nurmela, K. J. and Östergárd, P. R. J., Packing up to 50 equal circles in a square, *Disc. Comput. Geom.*, 18, 111, 1997.

[61] Edelsbrunner, H., Guibas, L. J., Pach, J., Pollack, R., Seidel, R., and Sharir, M., Arrangements of curves in the plane: topology, combinatorics, and algorithms, *Theor. Comp. Sci.*, 92, 319, 1992.

[62] Amato, N. M., Goodrich, M. T., and Ramos, E. A., Computing the arrangement of curve segments: divide-and-conquer algorithms via sampling, *Proc. SODA*, 2000, p. 705.

[63] Keil, M. and Snoeyink, J., On the time bound for convex decomposition of simple polygons, *Int. J. Comput. Geom. Appl.*, 12(3), 181–192, 2002.

[64] Wang, J., Medial axis and optimal locations for min-max sphere packing, *J. Comb. Opt.*, 4, 487–503, 2000.

79

Large-Scale Global Placement

Jason Cong
University of California

Joseph R. Shinnerl
Tabula, Inc.

Nearly five decades of steady exponential improvement in the design and manufacture of very large-scale integrated circuits (VLSI) has produced some of the largest combinatorial optimization problems ever considered. *Placement*—arranging the elements of a circuit in the plane—is one of the most difficult of these. As of 2007, mixed integer nonconvex nonlinear-programming formulations of placement with over 10 million variables and constraints are not unusual, and problem sizes continue to grow with Moore's law. Realistic objectives and constraints for placement incorporate complex models of signal timing, power consumption, wiring routability, manufacturability, noise, temperature, and so on. A popular and very useful simplification is to minimize a standard estimate of total wirelength subject only to pairwise nonoverlap constraints. Although this abstract model problem cannot fully express the scope of the placement challenge, evidence suggests that it does capture a critical part of the core mathematical difficulty.

In 1976, Sahni and Gonzalez [1] showed that, unless $P = NP$, no deterministic polynomial-time approximation algorithms for placement exist. In practice, problem sizes and available computing resources prohibit any order of run time beyond approximately $N \log N$, where N is the number of movable objects. Despite these obstacles, competition continues to push researchers toward a clearer understanding of the achievable limits of scalable algorithms.

This chapter presents an overview of metaheuristics for *global* placement as defined below. Section 79.1 contains a brief description of the problem's context and typical abstract formulation. In Section 79.2, dominant placement metaheuristics are reviewed. In Section 79.3, a brief overview is given of formulations modeling timing and routability. Conclusions are drawn in Section 79.4.

79.1 Background

Integrated circuit (IC) design consists of three main stages: behavioral, logical, and physical. The behavioral and logical stages translate the desired functionality of the IC into a directed graph. The nodes of the graph represent functional or storage components. The edges of the graph express signal paths, each signal constrained to arrive within prescribed time intervals at the nodes along its path from primary input to primary output. The task of *physical design* is to compute a *layout,* that is, spatial positions for all circuit elements and their interconnecting wires, consistent with the logical design. Physical design is usually further divided into separate placement and routing phases. During *placement,* the circuit's modules are arranged without overlap within a two-dimensional rectangular region of prescribed dimensions. After placement, required connections among modules are explicitly constructed during *routing* as spatially disjoint wiring paths in parallel planes over the modules.

The exponential growth of on-chip complexity has dramatically increased the demand for scalable optimization algorithms for large-scale physical design. Although complex logic functions are usually composed hierarchically, studies (e.g., [2]) show the importance of building a good physical hierarchy from a flattened or nearly flattened logical netlist for performance optimization. Because a logical hierarchy is usually conceived with little or no consideration of the layout and interconnect information, it may not map well to a two-dimensional layout. Therefore, large-scale global placement on a nearly flattened netlist is needed for physical hierarchy generation to achieve the best performance.

This approach is even more important in today's nanometer designs, where the interconnect has become the performance bottleneck. Interconnect delay grows roughly linearly with the insertion of buffers. However, as module sizes have decreased, their internal signal delays have become small compared to the delays of the wires joining them, especially when the modules are far apart. Placement determines the interconnect structure and, hence, the performance of the resulting circuit more than any other step in the VLSI design sequence. Thus, the continued exponential decrease of circuit element sizes has increased the relative importance of placement in the design flow.

79.1.1 Mathematical Formulation

An instance of the VLSI placement problem is specified as a hypergraph netlist $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, which is the output of logic synthesis. The terminology is illustrated in Figure 79.1. The vertices $v_i \in \mathcal{V}$ of $\mathcal{H}$ are rectangular modules of prescribed functionality. Each hyperedge or *net* $e \in \mathcal{E}$ is defined as a subset of the vertices, $e \subset \mathcal{V}$. Placement traditionally consists of two stages: global placement and detailed placement. In global placement, approximate locations of all modules are computed under relaxed formulations of design constraints. A strictly legal configuration is then computed prior to detailed placement, which maintains strict feasibility at every step.

During global placement, it is customary to project the modules $v \in \mathcal{V}$ into two dimensions ("2-D") and consider only their *widths* along the x-direction and *heights* along the orthogonal y-direction. Most modules are *standard cells* selected from a given library during *technology mapping*. Such cells consist of up to several dozen logic gates[1] each and have fixed dimensions. Cell widths are various integer multiples of unit length, but every cell has the same, fixed, uniform height. The placement region $\mathcal{R}$ has fixed rectangular shape and is correspondingly partitioned into uniform rows, the height of which matches the standard cell height. Increased design complexity has brought ever greater reuse of larger

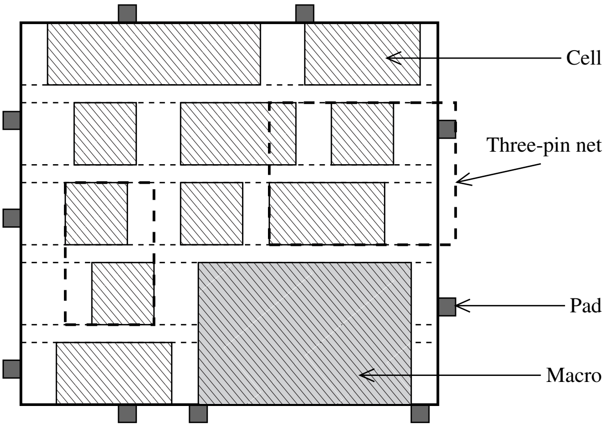


FIGURE 79.1 A sample placement illustrating basic terminology. Cells and macro (lower right) are shaded. Two nets are shown as broken lines enclosing subsets of cells and pads.

[1] For example, NAND, XOR, or NOT gates, often with different driving strengths.

intellectual property (IP) blocks called *macros*, which may be thousands of times as large as the standard cells and thus span several standard rows. The multiplicity and diversity of macro sizes in large-scale mixed-size placement presents a particular challenge to the development of robust, high-quality heuristics.

Let $(x_i,\ y_i)$ denote the coordinates of the lower-left corner of module i, and let $(w_i,\ h_i)$ denote its width and height. Let x and y denote corresponding vectors of the x_i and y_i. An accurate calculation of routed wirelength requires both detailed modeling of 3-D routing topologies and an actual routing solution. For simplicity, a nonconstructive[2] 2-D-wirelength approximation is used instead as a substitute. The bounding-box wirelength of a single net $e \in \mathcal{E}$ is simply half the perimeter of its smallest circumscribing rectangle,

$$w(e) = \max_{v_j \in e}(x_j + w_j) - \min_{v_k \in e} x_k + \max_{v_m \in e}(y_m + h_m) - \min_{v_n \in e} y_n \tag{79.1}$$

The corresponding weighted half-perimeter wirelength (HPWL) for a given placement is thus

$$f(x, y) = \sum_{e \in E} \gamma(e) w(e) \tag{79.2}$$

where the weights $\gamma(e)$ may be chosen adaptively for various purposes, for example, dynamically prioritizing nets by timing criticality (Section 79.3). In general, several of the modules' locations will be fixed a priori, but the number of fixed modules is a small fraction $(\sim 1/\sqrt{N})$ of the total.

Given the shapes of the modules and a specification of their hypergraph netlist $\mathcal{H}$, a precise formulation of placement constrains all modules to lie in standard cell rows (macros span multiple rows) with no two modules overlapping; this view amounts to a mixed integer nonlinear-programming problem. What distinguishes global placement from general and detailed placement is its approximation of the nonoverlap constraints. Typically, this approximation is expressed simply as a collection of simple constant upper bounds u_{ij} of module areas in subregions ("bins") B_{ij} defined by a regular $m \times n$ rectangular grid G. Thus, the global placement model problem may be expressed as

$$\begin{array}{ll} \min\limits_{x,y} & \sum_{e \in E} \gamma(e) w(e) \\ \text{subject to} & \sum_{v \in \mathcal{V}} \text{area}\ (v \cap B_{ij}) \le u_{ij} \quad \text{for all } i \in \{1, \ldots, m\},\ \ j \in \{1, \ldots, n\} \end{array} \tag{79.3}$$

where v and B_{ij} are viewed in this formula as sets of points in the plane. The resolution of the grid is usually determined by some empirical estimate of the capabilities and limitations of the legalization and detailed placement steps that follow. Illustrations of a global placement and a corresponding detailed placement, both produced by mPL5 [3] on a mixed-size circuit with over one million modules and nets, is shown in Figure 79.2.

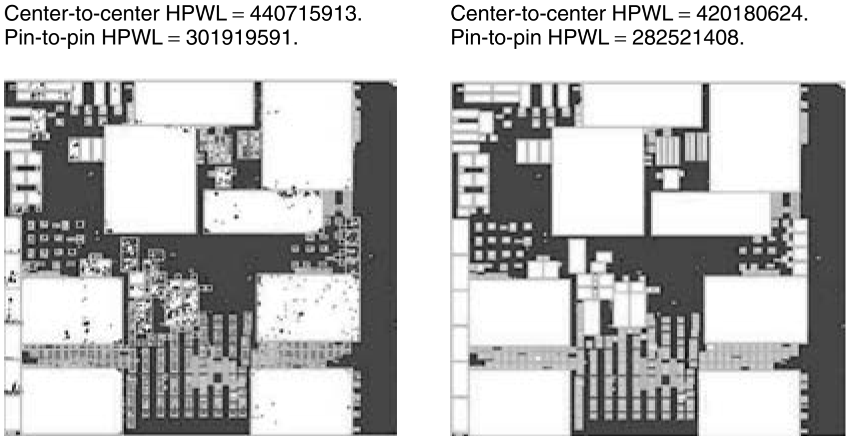

Center-to-center HPWL = 440715913.
Pin-to-pin HPWL = 301919591.
Center-to-center HPWL = 420180624.
Pin-to-pin HPWL = 282521408.

FIGURE 79.2 A global placement (left) with a corresponding detailed placement on a circuit with over one million movable objects. For simplicity, nets are not shown.

[2] The approximation is nonconstructive in the sense that no explicit physical signal routes are actually calculated.

While there is general agreement on formulation (Eq. [79.3]) among active researchers, the actual formulation of global placement used, if it is formally stated at all, is usually tailored ad hoc to suit a given algorithm. New variations continue to be investigated. For example, Eq. (79.3) is often viewed as a discretization of continuous area–density function $d(\bar{x}, \bar{y})$ defined at every point $(\bar{x}, \bar{y}) \in \mathcal{R}$. Extensions or alternatives which incorporate more detailed modeling of complex objectives and constraints, such as routability, signal propagation times, maximum temperature, noise, etc., are crucial in practice.

79.2 Overview of Dominant Metaheuristics

Heuristics for placement may be broadly classified as either hierarchical or flat. Prior to the 1980s, most research focused on flat heuristics for instances up to at most a few thousand modules and nets [4]. With the explosion in instance sizes due to Moore's law, most research has shifted to hierarchical frameworks supporting fast and scalable implementations [5]. Generally, however, the two views are combined in various ways. Flat heuristics typically play an enabling role at each level of a hierarchical algorithm. Conversely, flat formulations may rely on hierarchical numerical schemes to accelerate their internal calculations.

79.2.1 Flat Improvement Heuristics

In their survey article of 1972, Hanan and Kurtzberg [4] divided placement techniques into three categories: (i) constructive initial placement; (ii) iterative placement improvement; and (iii) branch and bound. Constructive initial placement incrementally selects and places unplaced movable modules according to the strength of their connectivity to already fixed modules (most circuits have at least some small subset of terminals fixed a priori). The process is simple and fast, but neglecting connections among movable modules diminishes the quality of the final placement. Branch and bound is far more accurate and also constructive but is affordable only for subproblems of ~15–20 modules or fewer.

Of these three early kinds of placement techniques, iterative improvement is the only one still widely used. In this approach, a given placement is repeatedly modified as long as sufficient reduction in the objectives is obtained. Early usage of the term is usually restricted to sequences of strictly *feasible,* that is, overlap-free, placements. While many iterative heuristics today also generate sequences of strictly feasible placements, other *infeasible* methods attain legality only approximately or asymptotically. Most feasible heuristics are either discrete or linear. Dominant infeasible heuristics include nonlinear, analytical formulations such as force-directed methods. Modification strategies in iterative improvement may be randomized or deterministic, localized or global.

79.2.1.1 Iterative Improvement over Feasible Placements

Perhaps, the simplest but least efficient placement procedure is the global *Monte Carlo* strategy attempted in early work [4]. In this approach, all modules are randomly assigned positions according to a given probability distribution. The resulting placement is retained if and only if it produces lower cost than previously obtained placements. Although the probability distribution can be dynamically adapted to push modules toward subregions likeliest to produce lower cost, results are not generally competitive. Currently, the most successful randomized algorithms employ sequences of local moves guided by *simulated annealing* [6–11]. A given placement is endowed with a neighborhood structure. A pair of neighboring modules is randomly selected, and the change in cost ΔC associated with exchanging the two modules' positions is computed. If $\Delta C < 0$, then the modules' positions are swapped—the move is accepted. If $\Delta C \geq 0$, then the move is accepted with probability proportional to $e^{-\Delta C/T}$, where T is the parameter simulating temperature. Initially, T is set large, so that *hill-climbing* moves are accepted with relatively high probability. Eventually, T is decreased far enough that such uphill moves are essentially excluded. When T is decreased sufficiently slowly, certain theoretical guarantees exist for asymptotic convergence of the process to a global optimum [90]. In practice, a far more rapid decrease in T must be used to keep run times acceptable. The main drawback of SA is its inherent lack of scalability. Genetic algorithms and simulated evolution have also been used in placement [12] but, to our knowledge, are not directly used by leading tools.

Early deterministic heuristics include techniques based on linear assignment, network flows, and force equilibration. In a typical assignment-based scheme, a subset of movable vertices is selected. If all movable vertices have the same dimensions, and no movable vertex shares a hyperedge with any other, then linear assignment (bipartite matching) can be used to determine an optimal permutation of the movable vertices over their set of locations. Construction of multiple subsets of nonadjacent vertices by iterative deletion is simple and fast on hypergraphs of bounded degree.

A generalization of this approach, called relaxation-based local search (RBLS), is introduced by Hur and Lillis [13,14]. In this scheme, the optimal locations of all vertices in a given movable subset are simultaneously determined without regard to overlap via solution of two separate rectilinear distance facility location (RDFL) subproblems, one for each coordinate direction. In the x-direction, the problem may be written as follows. Let $M \subset E$ denote the set of nets containing movable vertices; M typically also contains many other, fixed vertices. Variables r_i and l_i are introduced to represent the right and left boundaries of the ith net $e_i \in M$.

$$\begin{array}{ll} \min & \sum_{\{e_i \in M\}} r_i - l_i \\ \text{s.t.} & l_i \leq x_j \leq r_i \quad \text{for all } v_j \in e_i \end{array}$$

This problem can be solved efficiently by either a sequence of related network flows or by a single network flow applied to its dual. Once the subproblem has been solved and the optimal locations are determined, cell swapping along monotone chains of bins (“ripple-move”) is used to restore area–density legality. For each overfull bin s, a nearest underfull bin t is selected, and a chain of cell swaps between neighboring bins $(a_i,\ a_{i+1})$ leading from $s = a_1$ to t is computed. The chain is monotone in the sense that the Manhattan distance between a_i and t strictly decreases with i; this property and memoization reduce computational overhead. Network flows have also been used extensively in legalization and detailed placement algorithms [15,16].

79.2.1.2 Iterative Improvement over Infeasible Placements

Contemporary approaches to iterative improvement fall mostly among the so-called analytical methods based on mathematical programming or the equilibration of simulated forces. Generally, these methods all use continuous approximation and seek to satisfy a set of computationally verifiable optimality conditions either repeatedly for sequences of subproblems or asymptotically for the entire circuit. In contrast to the methods described in the previous subsection, they do *not* normally terminate at overlap-free configurations, and they therefore require postprocessing by a legalization engine.

The idea of modeling an IC as a system of springs and masses dates back at least as far as 1967 [17]. The mass of a vertex is taken in proportion to its area. The force on a mass i due to mass j is defined by Hooke’s law: $F_{ij} = k_{ij}s_{ij}$. Vector s_{ij} is the displacement from the position of i to that of j. Spring constant k_{ij} is proportional to the total relative strength of all hyperedges containing both i and j. The precise form of k_{ij} amounts to a prescription for approximating netlist H by a graph G; for example, $k_{ij} = \sum_{i,\, j \in e} w(e)/(|e| - 1)$. One simple form of iteration attempts to move vertices one by one to the available location nearest where the sum of the forces on them is zero. Later work of Quinn and Breuer [18] applies a Newton-based algorithm to a simultaneous systems of nonlinear equations for force equilibrium.

Other abstractions of force simulations have been proposed for placement. For example, Cheng and Kuh [19] proceed by an analogy to the minimization of power dissipation in an electrical network. More generally, explicit modeling of physical forces can be abandoned in favor of a simple mathematical model in which a quadratic, graph-based-wirelength approximation is minimized without regard to overlap constraints. This simple unconstrained quadratic placement is widely used to generate both initial placements and subsequent refinements, both local and global. The presence of fixed terminals at various locations tends to spread cells enough that cell centers rarely coincide, thus enabling subsequent spreading based on relative-order heuristics [20,21]. Judicious iterative addition and adjustment of fixed pseudoterminals is used by FastPlace [22] and mFAR [23,24] both for accelerating convergence and improving quality.

Example: Kraftwerk

Seminal work by Eisenmann and Johannes [25] formulates force-directed placement as a sequence of unconstrained quadratic minimizations. The perturbed quadratic-wirelength-objective function

$$q(x, y) = \frac{1}{2}(x^T Qx + y^T Qy) + b_x^T x + b_y^T y + f_x^T x + f_y^T y$$

captures both netlist connectivity and area congestion by a graph approximation and force–field calculation, as follows. Cell-to-cell connections determine the off-diagonal entries and part of the diagonal entries in the fixed graph Laplacian matrix Q by means of a quadratic star-wirelength model [20]. Cell-to-pad connections contribute to the diagonal elements of Q, rendering it positive definite, and determine the linear-term coefficients in the right-hand side vector $b = (b_x, b_y)$. Viewing this vector b as external spring-like forces following Hooke's law, the circuit connectivity is represented by the (constant) symmetric positive-definite matrix Q and the vector b. The perturbation vector $f = (f_x, f_y)$ represents global area-distribution forces analogous to electrostatic repulsion, with cell area playing the role of electric charge. At each iteration, vector f is recalculated from the current cell positions by means of a fast Poisson-equation solver. Since Q does not change from one iteration to the next unless nets are reweighted, a hierarchical set of approximations to Q can typically be reused over several iterations. We refer to this approach as *Poisson-based.*

79.2.2 Hierarchical Methods

The FastPlace and Kraftwerk algorithms show that global placement can still be done flat. Most other leading algorithms, however, explicitly incorporate hierarchy. The dominant hierarchical metaheuristics are recursive partitioning and multilevel methods, aka multiscale methods.

79.2.3 Recursive Partitioning

Among academic placement tools, all the leading top-down methods rely on variants of recursive circuit partitioning in some way. Seminal work on partitioning-based placement was done by Breuer [26] and Dunlop and Kernighan [27]. Most contemporary methods have exploited further advances in fast multiscale algorithms for hypergraph partitioning [28–30] to push these frameworks beyond their original capabilities. Fast, high-quality $\mathcal{O}(N)$ partitioning algorithms give top-down partitioning attractive $\mathcal{O}(N \log N)$ scalability overall. The asymptotic is $\mathcal{O}(N \log N)$ and not $\mathcal{O}(N)$, because partitioning is always applied to cells, not to aggregates.

79.2.3.1 Cutsize Minimization

At a given level of the top-down hierarchy, each rectangular subregion $\mathcal{S}$ and the modules assigned to it are bipartitioned, that is, $\mathcal{S}$ is split by a horizontal or vertical *cutline* into two disjoint rectangular subregions $\mathcal{S}_1$ and $\mathcal{S}_2$, and each module assigned to $\mathcal{S}$ is assigned to either $\mathcal{S}_1$ or $\mathcal{S}_2$. Most partitioning-based top-down placers employ variations of multilevel [28–30] Fiduccia-Matheysses (FM) style [31] iterations to separate the modules. Given some initial partition, subsets of cells are moved across its cutline in a way that reduces the total weight of hyperedges cut without violating a given area–balance constraint (a hyperedge is *cut* if it contains modules in both subsets of the partition). Leading tools based on recursive cutsize-driven partitioning include Capo [32,33] and Feng Shui [34,35]. Spatial cutlines for subregions, either horizontal or vertical, can be carefully chosen, for example, by dynamic programming [36], such that subregion aspect ratios remain bounded. As the recursion proceeds, cell subsets become smaller, and the cell-area distribution over the placement region becomes more uniform. Base cases of the bipartitioning recursion are reached when cell subsets become small enough that special end-case placers can be applied [37]. A small example is illustrated after three levels of bipartitioning in Figure 79.3.

Netlist bipartitioning can be enhanced in a few important ways to support the ultimate goal of wirelength-driven circuit placement. Key considerations include (i) terminal propagation; (ii) subproblem

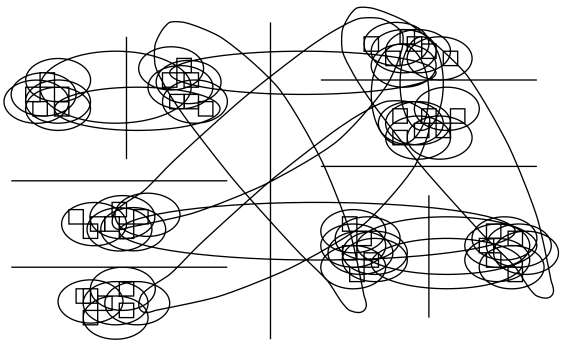


FIGURE 79.3 Cutsize-driven partitioning-based placement. Rectangles represent movable cells, line segments represent cutlines, and ellipses and other closed curves represent nets. The recursive bipartitioning attempts to minimize the number of nets containing cells in more than one subregion.

ordering; (iii) cutline placement; and (iv) handling small balance tolerances and/or highly nonuniform module areas.

Connections between subregions can be modeled by *terminal propagation* [27,38], in which the usual cutsize objective is augmented by terms incorporating the effect of connections to external subregions. At early stages, when the best positions of these connection points are not clear, several iterations may be used to incorporate feedback [34,39]. Other techniques for organizing local partitioning subproblems use Rent's rule to relate cutsize to wirelength estimation [40,36].

Careful consideration of the order and manner in which subregions are selected for partitioning can be significant. In the *multiway partitioning* framework, intermediate results from the partitioning of each subregion are used to influence the final partitioning of others. Explicit use of multiway partitioning at each stage can in some cases bring the configuration closer to a global optimum than is possible by recursive bisection alone [34]. Cell replication and iterative deletion have been used for this purpose [41]. Rather than an attempt to find the best subregion in which to place a cell, one can replicate the cell enough times to place it once in every subregion, then iteratively delete only the worst choices. These iterations may continue until only one choice remains, or they may be terminated earlier, allowing a small pool of candidates to be propagated to and replicated at finer levels. By postponing further deletion decisions until better information becomes available, spurious effects from locally optimal subregion partitions can be diminished and the global result improved.

In Capo, horizontal cuts are constrained to lie between uniform-height rows of the standard-cell layout. Respecting standard-cell row boundaries in this fashion greatly facilitates legalization of the final global placement. However, a recent study shows [42] that this restriction occasionally overconstrains end cases and increases wirelength. The authors of this study show that Feng Shui's "fractional-cut" relaxation of row boundaries during the partitioning can considerably improve results, when it is followed by careful displacement-minimizing legalization, such as dynamic-programming-based row assignment.

Much of Capo's performance derives from its placement-driven enhancements to its core FM partitioner [43,44] to support nonuniform module sizes and tight area–balance constraints. Given any initial partition, FM considers sequences of single, maximum-gain cell moves from one partition block to the other. It maintains a list of "buckets" for each partition block, where the *k*th bucket in each list holds the vertices which, when moved to the opposite block, will reduce the total number of nets cut by *k*. However, a cell will not be moved if the move violates the vertex-weight (area) balance constraint. A large module in an FM gain bucket must not prevent other modules of equal gain from being considered for movement. Capo starts each bipartitioning subproblem with a relaxed area–balance constraint and gradually tightens the constraint as partitioning iterations proceed. As the balance tolerance decreases below the area of any cell, that cell is locked in its current partition block. The final subproblem balance tolerance is selected so that, given an initial white space budget, enough relative white space in end-case subproblems is ensured so that overlap-free configurations can typically be found.

Incorporating Advances in Floorplanning

The quality and robustness of partitioning-based placement, especially mixed-size placement, can be enhanced by the incorporation of techniques for fixed outline floorplanning. Such incorporation improves the handling of large macro blocks [45] and facilitates final legalization of end-case subproblems. Alternative approaches are taken by Capo [32,33] and Patoma/PolarBear [46,47].

In Capo [32,33], min-cut placement proceeds as described above until certain ad hoc tests suggest that legalization of a subset of macro blocks and cells within their assigned subregion may be difficult. At that point, the cells in that subregion are aggregated into soft clusters, and annealing-based fixed outline floorplanning is applied to the given subproblem [48]. If it succeeds, the macro locations in its solution are fixed. If it fails, it must be merged with its sibling subproblem, and the merged parent subproblem must then be floorplanned. This step therefore initiates a recursive backtracking through ever larger ancestor subproblems. The backtracking terminates when one of these ancestor subproblems is successfully floorplanned. The ad hoc tests are chosen to prevent long backtracking sequences on most cases.

The challenge of ensuring the legalizability of subproblems within a min-cut partitioning-based floorplanning or placement has been most directly addressed by Patoma and PolarBear [47,49].[3] Beginning with the given instance itself, PolarBear employs fast and scalable area-driven floorplanning before cutsize-driven partitioning to confirm that the problem can be legalized as given. This area-driven "prelegalization" ignores wirelength but serves as a guarantor of the legalizability of subsequent steps. It is extremely robust. Given the guarantor legalization at a given level, cutsize-driven partitioning proceeds at that level. The flow then proceeds recursively on the subproblems generated by the cutsize-driven partitioning, each subproblem being legalized before it is solved. When prelegalization fails, the failed subproblem is merged with its sibling, and the previously computed legal guarantor solution to this parent subproblem is improved to reduce wirelength. The flow thus guarantees the computation of a legal placement or floorplan, under the very modest assumption that the initial attempt to prelegalize the given instance succeeds. Experiments with this flow demonstrate significantly more robust performance on mixed-size benchmarks with white space between 1 and 10%.

79.2.3.2 Partitions Guided by Analytical Placements

An oft-cited disadvantage of recursive bisection is its alleged tendency to ignore the global objective as it pursues locally optimal partitions. Approximating wirelength by cutsize in the objective may also degrade the quality of the final placement. A radically different approach, first introduced in Proud [19,50], and subsequently, refined by Gordian [20,51], BonnPlace [21,52], and Warp [53], is to use continuous, iteratively constrained quadratic star-model-wirelength minimization over the entire circuit to guide partitioning decisions. The choice of a quadratic-wirelength objective helps avoid long wires and facilitates the construction of efficient numerical linear-system solvers for the optimality conditions, for example, preconditioned conjugate gradients. Fixed I/O terminals prevent modules from simply collapsing to a single point. Linear wirelength can still be asymptotically approximated by iterative adjustments to the net weights [51].

Following this "analytical" placement, each region is then quadrisected, and cells are partitioned to subregions to further reduce overlap and area congestion. In Gordian, carefully chosen cutlines and FM-based cutsize-driven partitioning and repartitioning are used. Cell-to-subregion assignments are loosely enforced by imposing and maintaining a single center-of-mass equality constraint for each subregion. As constraints accumulate geometrically, degrees of freedom in cell movement are eliminated, and the quadratic minimization at each step moves cells less and less. In BonnPlace, module subsets are quadrisected in a manner that essentially minimizes the sum of their rectilinear displacements from their starting positions [54]. BonnPlace does not explicitly impose equality constraints into the subsequent analytical minimization to preserve these partitioning assignments, as Gordian does. Instead, it directly alters the quadratic-wirelength objective to minimize the sum of all cells' squared Manhattan displacements from their assigned subregions.

[3] Recently [91], ad-hoc look-ahead floorplanning based on simulated annealing of large macros with clustered smaller macros and standard cells has been incorporated into Capo with excellent results.

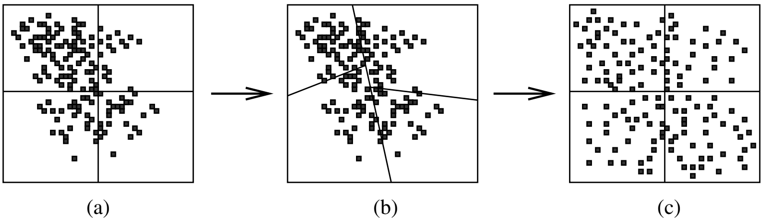


FIGURE 79.4 Warping. Each bin of a uniform bin grid (a) is mapped to a corresponding quadrilateral in an oblique but slicing bin structure (b) so as to capture roughly equal numbers of cells in each quadrilateral. The inverse bin maps are applied to the cells in order to spread them out (c).

The novelty of grid warping [53] is that, rather than directly move modules based on their distribution, it uses the modules to deform or *warp* the region in which they lie, in an analogy with gravity as described by Einstein's general relativity. The inverse of the deformation is then used to carry modules from their original locations to a more uniform distribution. Figure 79.4 illustrates the approach. As shown, oblique grid lines are used, and although a slicing structure with alternating cutline directions and quadrilateral bins is maintained, gridlines not necessary to the slicing pattern are broken at points where they intersect other gridlines. This weakening of the grid structure allows close neighbors in the original unconstrained placement to be separated a relatively large distance by the warping. The grid points of the warped grid are determined simultaneously by a derivative-free method of nonlinear optimization of Brent and Powell [55]. The required top-down slicing grid structure is maintained by (a) fixing the alternating cutline direction order a priori, by deciding whether to orient the first cut from top to bottom or from side to side, and (b) expressing each cutline after the first in terms of two variables, one for where it intersects its parent cutline, and another for where it intersects the opposite boundary or cutline. A penalty function f is used as the objective:

$$f = \text{wirelength} + \rho \sum_{\text{bins}} \beta_{ij}$$

where β_{ij} is approximately the square of the difference between the total cell area in bin (i, j) and the target cell area $\kappa = \kappa(i, j)$ for each bin. The wirelength is the total weighted HPWL obtained after the inverse warp. Although evaluating the objective is fairly costly, the number of variables in the optimization is low—only 6 for a 2×2 grid or 30 for a 4×4 grid—and convergence is fast.

Iterative Refinement

Following the initial partitioning at a given level, various means of further improving the result at that level can be used. In BonnPlace (Section 79.2.3.2), unconstrained quadratic- wirelength minimization over 2×2 windows of subregions is followed by a repartitioning of the cells in these windows. Windows can be selected based on routing–congestion estimates. Capo [32] greedily selects cell orientations to reduce wirelength and improve routability. Feng Shui [34] follows k-way partitioning by localized repartitioning of each subregion. Some leading partitioning-based placers also employ time-limited branch-and-bound-based enumeration at the finest levels [37].

In Dragon [10,40], an initial cutsize-minimizing quadrisection is followed by a bin-swapping-based refinement, in which entire partition blocks at that level are interchanged in an effort to reduce total wirelength. At all levels except the last, low-temperature simulated annealing is used; at the finest level, a more detailed and greedy strategy is employed. Because the refinement is performed on aggregates of cells rather than on cells from the original netlist, Dragon may also be grouped with the multilevel methods discussed next.

79.2.4 Multiscale Methods

Placement algorithms in the multilevel paradigm have developed rapidly over the last 6 years [3,10,11,24, 40,56–62]. These methods construct a hierarchy of problem approximations, perform optimizations on aggregated variables and data functions at each level, and transfer solutions between levels to obtain a final placement of the given netlist. The following terminologies are standard.

(i) *Coarsening.* Hierarchies are built recursively, by bottom-up aggregation or top-down partitioning.
(ii) *Relaxation.* Iterative optimization improves an approximate solution at a given aggregation level.
(iii) *Interpolation.* Each final approximate placement at a given level is used as the initial placement at its neighboring finer level. (A good placement for the coarsest level can be obtained directly.)

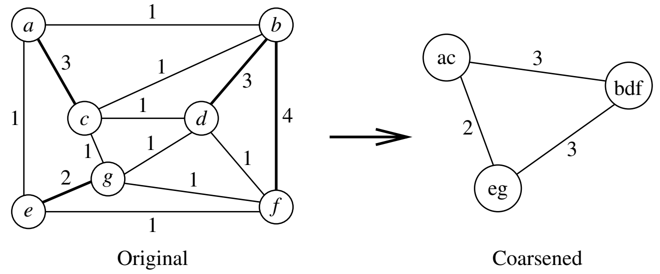
The order in which the various problems at the various levels are solved can also be important. The simplest and most common approach is simply to proceed top down, from the coarsest to the finest level, once the aggregation hierarchy has been constructed [10,56,57,60]. However, studies show that considerable improvement is possible by repeated traversals and reconstructions of the hierarchy in various orderings [59,63], as in traditional multiscale methods for Partial Differential Equations (PDEs) [64]. We refer to this organization of traversals as *iteration flow.*

The scalability of the multilevel approach is obvious. Provided relaxation at each level has order linear in the number N_a of aggregates at that level, and the number of aggregates per level decreases by factor $r < 1$ at each level of coarsening, say $N_a(i) = r^i N$ at level i, the total order of a multilevel method is at most $cN(1 + r + r^2 + \cdots) = cN/(1 - r)$. Higher-order (nonlinear) relaxations can still be used in a limited way (e.g., Refs. [3,24,60]). For example, a hard limit on the number of global nonlinear relaxation steps can be imposed. While not strictly scalable, global relaxations often produce better solutions than their localized counterparts and can be tuned to limit run time. Alternatively, relaxation can be applied only to subsets of bounded size, for example, by sweeps over overlapping windows of contiguous clusters at the current aggregation level.

79.2.4.1 Coarsening

Traditional multiscale algorithms form their hierarchies by recursive clustering or generalizations thereof. However, the importance of limiting cutsize makes partitioning attractive in the placement context [10]. Typically, clustering algorithms merge tightly connected cells in a way that eliminates as many nets at the adjacent coarser level as possible while respecting some area–balance constraints. Experiments to date suggest that relatively simple, graph-based greedy strategies like First-Choice vertex matching [65,66] produce fairly good results. More sophisticated ideas like edge-separability clustering (ESC) [67], wirelength-prediction-based clustering [68], and "best-choice" clustering [66] have also been attempted. In general, how best to define coarse-level hyperedges without explosive growth in the number and degree of coarsened hyperedges relative to coarsened vertices remains an important open question [69].

First-Choice clustering is illustrated in Figure 79.5. A graph is defined on the netlist vertices with each edge weighted by the "affinity" of the given two vertices. The affinity may represent some weighted combination of complex objectives, such as hypergraph connectivity, spatial proximity, timing delay, area



Original

Coarsened

FIGURE 79.5 First-Choice clustering on an affinity graph. Darkened edges in the original graph are of maximal weight for at least one of their vertices. Note that vertex d has maximal affinity for vertex b, but vertex b has maximal affinity for vertex f.

balance, coarse-level hyperedge elimination, and so on. Each vertex is paired with some other vertex for which it has its highest affinity. This maximum-affinity pairing is not symmetric and is independent of the order in which vertices are considered (see Figure 79.5). The corresponding maximum-affinity edges are marked and define a subgraph of the affinity graph; connected components of this subgraph are clustered and thus define vertices at the next coarser level.

79.2.4.2 Initial Placement at Coarsest Level

A placement at the coarsest aggregate level may be derived in various ways. Because the initial placement may have a large influence at subsequent iterations, and because the coarsest-level problem is relatively small, the placement at this level is typically performed with great care, to the highest quality possible, by an enhanced version of the relaxation engine. How to judge the coarse-level placement quality is not necessarily obvious, however, as the coarse-level objective may not correlate strictly with the ultimate fine-level objectives.

79.2.4.3 Relaxations

The core of a multiscale algorithm is the means by which it improves its approximate solution at a given aggregation level. Almost any algorithm for intralevel optimization can be used, provided that it can support (i) incorporation of complex constraints and (ii) restriction to subsets of movable objects. Leading contemporary algorithms build on the iterative improvement heuristics employed by their predecessors. Relaxation in mPG [11], Dragon [10], and Ultrafast VPR [56] is by fast annealing. Both APlace [60,61] and mPL5 [92] use nonlinear programming with the following log–sum–exp smoothing of the HPWL of each net $t = \{(x_i, y_i) \mid i = 1, \ldots, \deg(t)\}$,

$$\ell_{\exp}(t) = \alpha \cdot \left(\ln\left(\sum e^{x_i/\alpha}\right) + \ln\left(\sum e^{-x_i/\alpha}\right) + \ln\left(\sum e^{y_i/\alpha}\right) + \ln\left(\sum e^{-y_i/\alpha}\right)\right) \qquad (79.4)$$

where α is the smoothing parameter. The formulations used by APlace and mPL5 for nonoverlap constraints are quite different however, as are the optimization engines used to find solutions. These are reviewed next.

Example: APlace

In APlace, the scalar potential field $\phi(x, y)$ used to generate area–density-balancing forces is defined as a sum over cells and bins as follows. For a single cell v at position (x_v, y_v) overlapping with a single bin b centered at (x_b, y_b), the potential is the bell-shaped function

$$\phi_v(b) = \alpha(v)\, p(|x_v - x_b|)\, p(|y_v - y_b|)$$

where $\alpha(v)$ is selected so that $\sum_{b \in G} \phi_v(b) = \text{area}(v)$, and

$$p(d) \equiv \begin{cases} 1 - 2d^2/r^2 & \text{if } 0 \le d \le r/2 \\ 2(d - r)^2/r^2 & \text{if } r/2 \le d \le r \end{cases} \qquad (79.5)$$

and r is the *radius* of the potential. The potential ϕ at any bin b is then defined as the sum of the potentials $\phi_v(b)$ for the individual cells overlapping with that bin. Let (X, Y) denote all positions of all cells in the placement region R. Let $|G|$ denote the total number of bins in grid G. Then the target potential for each bin is simply $\bar{\phi} = \sum_{v \in V} \text{area}(v)/|G|$, and the area–density penalty term for a current placement (X, Y) on grid G is defined as

$$\psi_G(X, Y) = \sum_{b \in G} (\phi(b) - \bar{\phi})^2$$

For the given area density grid G, APlace then formulates placement as the unconstrained minimization problem

$$\min_{v \in V} \rho_\ell \left(\sum_{e \in E} \ell_{\exp}(e)\right) + \rho_\psi \psi_G(X, Y)$$

for appropriate, grid-dependent scalar weights ρ_ℓ and ρ_ψ. This formulation has been successfully augmented in APlace to model routing congestion, movable I/O pads, and symmetry constraints on placed objects.

Optimization in APlace proceeds by the Polak-Ribiere variant of nonlinear conjugate gradients [70] with Golden-Section linesearch [71]. A hard iteration limit of 100 is imposed. The grid size $|G|$, objective weights ρ_ℓ and ρ_ψ, wirelength smoothing parameter α (Eq. [79.4]), and area–density potential radius r (Eq. [79.5]) are selected and adjusted at each level to guide the convergence. Bin size and α are taken proportional to the average aggregate size at the current level. The potential radius r is set to 2 on most grids but is increased to 4 at the finest grid to prevent oscillations in the maximum cell-area density of any bin. The potential weight ρ_ψ is fixed at one. The wirelength weight ρ_ℓ is initially set rather large and is subsequently decreased by 0.5 to escape from local minima with too much overlap. As iterations proceed, the relative weight of the area–density penalty increases, and a relatively uniform cell-area distribution is obtained.

Example: mPL5

mPL5 generalizes the Kraftwerk framework to a more rigorous mathematical formulation suitable for a multilevel implementation. Recall that x and y denote vectors of module coordinates; hence, we let $(\bar{x}, \bar{y})$ denote an arbitrary point in $\mathcal{R}$. Letting D_{ij} denote the cell-area density of bin B_{ij} and K the total cell area divided by the total placement area, the area–density constraints are initially expressed simply as $D_{ij} = K$ over all bins B_{ij}. Viewing the D_{ij} as a discretization of the smooth density function $d(\bar{x}, \bar{y})$, these constraints are smoothed by approximating d by the solution ψ to the Helmholtz equation

$$\begin{aligned} \Delta\psi(\bar{x}, \bar{y}) - \epsilon\psi(\bar{x}, \bar{y}) &= d(\bar{x}, \bar{y}), \quad (\bar{x}, \bar{y}) \in \mathcal{R} \\ \frac{\partial\psi}{\partial\nu} &= 0, \quad (\bar{x}, \bar{y}) \in \partial\mathcal{R} \end{aligned} \tag{79.6}$$

where $\epsilon > 0$, ν is the outer unit normal, $\partial\mathcal{R}$ the boundary of the placement region $\mathcal{R}$, $d(\bar{x}, \bar{y})$ the continuous density function at a point $(\bar{x}, \bar{y}) \in \mathcal{R}$, and Δ the Laplacian operator $\Delta \equiv \frac{\partial^2}{\partial \bar{x}^2} + \frac{\partial^2}{\partial \bar{y}^2}$. The smoothing operator $\Delta_\epsilon^{-1} d(\bar{x}, \bar{y})$ defined by solving Eq. (79.6) is well defined, because Eq. (79.6) has a unique solution for any $\epsilon > 0$. Since the solution of Eq. (79.6) has two more derivatives [72] than $d(\bar{x}, \bar{y})$, ψ is a smoothed version of d. Discretized versions of Eq. (79.6) can be solved rapidly by fast numerical multilevel methods. Recasting the density constraints as a discretization of ψ gives the nonlinear programming problem

$$\begin{aligned} \min \quad & W(x, y) \\ \text{s.t.} \quad & \psi_{ij} = -K/\epsilon, \quad 1 \le i \le m, 1 \le j \le n \end{aligned} \tag{79.7}$$

where the ψ_{ij} is obtained by solving Eq. (79.6) with the discretization defined by the given bin grid. Interpolation from the adjacent coarser level defines a starting point. This nonlinear-programming problem is solved by the Uzawa iterative algorithm [73], which does not require second derivatives or large linear-system solves:

$$\begin{aligned} \nabla W(x^{k+1}, y^{k+1}) + \sum_{i,j} \lambda_{ij}^k \nabla\psi_{ij} &= 0 \\ \lambda_{ij}^{k+1} &= \lambda_{ij}^k + \alpha(\psi_{ij} + \bar{K}/\epsilon) \end{aligned} \tag{79.8}$$

where λ is the Lagrange multiplier, $\lambda^0 = 0$, α is a parameter to control the rate of convergence, and gradients of ψ_{ij} are approximated by simple forward finite differences $\nabla_{\bar{x}_k}\psi_{ij} = \frac{\psi_{i,j+1} - \psi_{i,j}}{h_{\bar{x}}}$, $\nabla_{\bar{y}_k}\psi_{ij} = \frac{\psi_{i+1,j} - \psi_{i,j}}{h_{\bar{y}}}$ when the center of cell v_k is inside B_{ij} and are set to zero otherwise. The nonlinear equation for (x^{k+1}, y^{k+1}) is recast as an ordinary differential equation and solved by an explicit Euler method [74].

Comparison to APlace

mPL5 and APlace both employ multilevel adaptations of globalized, analytical, iterative, formulations for placement. The primary difference between their formulations is the manner in which they model the nonoverlap constraints. APlace uses *local* smoothing of area densities derived from symmetric bell-shaped functions, while mPL5 uses a *global* smoothing derived from the Helmholtz equation. While mPL5 specifically targets first-order *constrained* optimality conditions with explicit Lagrange-multiplier updates derived from fast ODE solves, APlace minimizes each member of a sequence of unconstrained penalty functions. The convergence theory for these sequential unconstrained methods [75] shows that they implicitly maintain Lagrange-multiplier estimates as well. Both methods rely on (i) empirical density estimates for termination criteria and (ii) empirically tuned parameters for control of the convergence rate, and both have produced results of very high quality.

79.2.4.4 Interpolation

Simple declustering and linear assignment can be effective [57]. With this approach, each component cluster is initially placed at the center of its parent's location. If an overlap-free configuration is needed, a uniform bin grid can be laid down, and clusters can be assigned to nearby bins or sets of bins. The complexity of this assignment can be reduced by first partitioning clusters into smaller windows, for example, of 500 clusters each. If clusters can be assumed to have uniform size, then fast linear assignment can be used. Otherwise, approximation heuristics are needed.

Under algebraic-multigrid-style (AMG)-weighted disaggregation, each finer-level cluster is initially placed at the weighted average of the positions of all coarser-level clusters with which its connection is sufficiently strong [58]. Finer-level connections can also be used: once a finer-level cluster is placed, it can be treated as a fixed, coarser-level cluster for the purpose of placing subsequent finer-level clusters.

79.3 Timing and Routability

Realistic constraints for VLSI placement typically enforce limits on maximum signal-propagation times and the routability of wires connecting modules. The formulation of these constraints and their incorporation within the generic wirelength-driven model problem are described briefly in this section.

79.3.1 Timing

Timing-driven placement algorithms fall into two categories: path-based and net-based. An accurate view of signal propagation in an IC is graph-based rather than hypergraph-based. In each net of a given IC, one vertex serves as signal source and the other vertice serve as sinks. A signal travels from its primary input (PI) to its primary output (PO) along a path of pairs of modules in nets. The performance of a circuit is determined by the longest delay of any of its signal paths. Path delay is extremely complex, however, as the number of paths grows exponentially with circuit size.

Two *path-based* formulations of timing-driven placement appear frequently. In the first, the maximum signal-propagation time along any path is used directly as the objective to be minimized. In the second, constraints on propagation times are imposed, and the minimum slack along any path is maximized. In either formulation, auxiliary variables representing *arrival times* at circuit nodes are explicitly introduced. In terms of arrival time $a(i)$ at pin i, timing constraints may be expressed as follows:

$$a(j) \geq a(i) + d(i, j) \quad \forall (i, j) \in G; \qquad a(j) \leq T \quad \forall j \in PO; \qquad a(i) = 0 \quad \forall i \in PI$$

where G denotes the timing graph, $d(i, j)$ the delay of timing arc (i, j) either as a constant for cell internal delay or as a function of cell locations, and T the target's longests path delay. Here we assume that the arrival time at all *PI* pins is zero and that all *PO* pins have the same delay targets. Simple changes can be made to the formula to accommodate more complex situations.

The advantage of path-based algorithms is their accurate timing view during the optimization procedure. However, they usually require substantial computation resources, and, in certain placement frameworks such as top-down partitioning, it is very difficult or infeasible to maintain an accurate view of global timing.

Net-based algorithms [25,76–78], in contrast, enforce path-based constraints only indirectly by means of net-length constraints or net weights. This information is fed to a weighted-wirelength-minimization-based placement engine to obtain a new placement with better timing. This new placement is then analyzed by a static analyzer, thus generating a new set of timing information to guide the next placement iteration. Usually this process must be repeated for a few iterations until no improvement can be made or until a certain iteration limit has been reached. Net-weighting-based approaches assign weights to nets based on (a) their timing criticality and (b) the number of paths sharing a net. A recently proposed algorithm [79] can properly scale the impact of all paths by their relative timing criticalities as measured by their slacks. Under certain conditions, this method is equivalent to enumerating all the paths in the circuit, counting their weights, and then distributing the weights to all edges in the circuit.

79.3.2 Routability

Because most wire routes go over the modules in parallel planes known as *routing layers,* during placement it is not strictly necessary to reserve space for wires alongside the modules. However, a tightly packed placement may be difficult or impossible to route. Therefore, quantitative models of estimated routing congestion are incorporated into the objective or constraints of practical placement algorithms.

There are two major categories of routability modeling: topology-free (TP-free), where no explicit routing is done and topology-based (TP-based), where rectilinear routing trees are explicitly constructed on some routing grid. TP-free modeling is faster in general. In bounding-box modeling [80], for example, the routing supply for each bin in the routing grid structure is modeled according to how the existing wiring of power or clock nets, regular cells, and macros is placed, and the routing demand of a net is modeled by its weighted bounding-box length. Pin densities and stochastic models for two-pin nets have also been used to compute expected horizontal and vertical track usage with consideration of routing blockages [81,82]. In TP-based modeling, for each net, a Steiner tree is generated on the given routing grid. If a TP-based modeling method uses a topology similar to what the after-placement-router does, the fidelity of the model can be guaranteed. However, topology generation is often of high complexity; therefore, most research focuses mainly on efficiency. In one approach [83], a precomputed Steiner tree topology on a few grid structures is used for wiring-demand estimation. In another approach [11], two algorithms of logarithmic complexity were recently proposed: a fast congestion-avoidance two-bend routing algorithm, LZ-router for two-pin nets, and an IncA-tree algorithm, which can support incremental updates for building a rectilinear Steiner arborescence tree (A-tree) for a multipin net.

The results of routability modeling can be applied to placement optimization by net weighting, cell weighting (cell inflation), or white-space allocation. Net weighting directly incorporates a congestion picture into the weighted-wirelength-placement objective. Cell weighting (also known as cell inflation) incorporates a congestion picture into nonoverlap constraints by inflating cell sizes based on congestion estimation, so that the placement algorithm's overlap-removal system can alleviate the congestion without added enhancement. White-space allocation can be applied hierarchically [93–95] to ease routing congestion in hot spots with low perturbation to a given layout. mPL6 [96], a recent enhancement of mPL5 [97], uses artificial, unconnected "filler cells" for efficient white-space distribution in the multiscale nonlinear-programming framework. mPL6 produced the best quality of results in the ISPD 2006 Placement Contest [98].

79.4 Conclusion

Recent years have seen a resurgence in the investigation of both synthetic benchmarks with known optimality properties [84–87] and lower bounds for globally optimal solutions to real instances [88]. For standard-cell benchmarks with known optima [85], leading tools may produce solutions with wirelengths as much as 2× or more above the optimal. For large mixed-size benchmarks with known optima [89], observed wirelengths are often 5× or more above the optimal. Moreover, this gap is observed to increase

20% or more with the size of the parametrized benchmarks. One recent study suggests that a large gap may result even when almost every module in a global placement is within a very small distance—one or two cell widths—of its optimal location [86]. While the results of these studies are diverse and difficult to generalize, the techniques they describe are increasingly being used to identify algorithm weaknesses that may be difficult to isolate on real instances.

In the last decade, the increased importance of interconnect delay on the performance of VLSI circuits has spurred a burst of progress in algorithms for large-scale global placement. The new algorithms often generalize earlier heuristics within a hierarchical framework—either top-down recursive partitioning or multiscale optimization. Increasing computing power and instance complexity continue to push researchers to consider diverse techniques as they seek more effective algorithms. More detailed discussion of recent advances in large-scale global placement can be found in [5] and [99].

Acknowledgment

Partial support for this work has been provided by Semiconductor Research Consortium Contract 2003-TJ-1091 and National Science Foundation Contract CCF 0430077. This chapter is derived from the article "Large-Scale Circuit Placement," ACM Transactions on Design Automation of Electronic Systems, 10(2), ISSN# 1084-4309 (April) Copyright ACM, 2005. http://doi.acm.org/10.1145/1059876.1059886

References

[1] Sahni, S. and Gonzalez, T., P-complete approximation problems, *JACM*, 23(3), 555, 1976.

[2] Cong, J., An interconnect-centric design flow for nanometer technologies, *Proc. IEEE*, 89(4), 2001, 505.

[3] Chan, T. F., Cong, J., and Sze, K., Multilevel generalized force-directed method for circuit placement, in *Proc. Int. Symp. on Physical Design,* 2005.

[4] Breuer, M. A., Ed., *Design Automation of Digital Systems: Theory and Techniques*, Vol. 1, Prentice-Hall, New Jersey, 1972, chap. 5.

[5] Cong, J., Shinnerl, J. R., Xie, M., Kong, T., and Yuan, X., Large-scale circuit placement, *ACM Trans. Des. Autom. Electron. Syst.*, 10(2), 389–430, 2005.

[6] Kirkpatrick, S., Gelatt, C. D., Jr., and Vecci, M. P., Optimization by simulated annealing, *Science*, 220, 671, 1983.

[7] Sechen, C., *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer Academic Publishers, Dordrecht, 1988.

[8] Sun, W.-J. and Sechen, C., Efficient and effective placement for very large circuits, *IEEE Trans. CAD*, 14(3), 349–359, 1995.

[9] Betz, V. and Rose, J., VPR: A new packing, placement, and routing tool for FPGA research, *Proc. Int. Workshop on FPL*, 1997, p. 213.

[10] Sarrafzadeh, M., Wang, M., and Yang, X., *Modern Placement Techniques*, Kluwer, Boston, 2002.

[11] Chang, C.-C., Cong, J., Pan, D., and Yuan, X., Multilevel global placement with congestion control, *IEEE Trans. CAD*, 22(4), 395, 2003.

[12] Sherwani, N., *Algorithms for VLSI Physical Design Automation*, 3rd ed., Kluwer Academic Publishers, Dordrecht, 1999.

[13] Hur, S.-W. and Lillis J., Relaxation and clustering in a local search framework: Application to linear placement, *Proc. ACM/IEEE Design Automation Conf.*, 1999, p. 360.

[14] Hur, S.-W. and Lillis, J., Mongrel: Hybrid techniques for standard-cell placement, *Proc. Int. Conf. on Computer Aided Design*, 2000, p. 165.

[15] Doll, K., Johannes, F. M., and Antreich, K. J., Iterative placement improvement by network flow methods, *IEEE Trans. CAD*, 13(10), 1189–1200, 1994.

[16] Brenner, U., Pauli, A., and Vygen, J., Almost optimum placement legalization by minimum cost flow and dynamic programming, *Proc. Int. Symp. on Physical Design*, 2004, p. 2.

[17] Fisk, C. J., Caskey, D. L., and West, L. L., Accel: automated circuit card etching layout, *Proc. IEEE*, 55(11), 1967, 1971.

[18] Quinn, N. and Breuer M., A force-directed component placement procedure for printed circuit boards, *IEEE Trans. CAS*, CAS-26, 377, 1979.

[19] Cheng, C.-K. and Kuh, E. S., Module placement based on resistive network optimization, *IEEE Trans. CAD*, CAD-3(3), 218–225, 1984.

[20] Kleinhans, J. M., Sigl, G., Johannes, F. M., and Antreich, K. J., GORDIAN: VLSI placement by quadratic programming and slicing optimization, *IEEE Trans. CAD*, 10, 356, 1991.

[21] Vygen, J., Algorithms for large-scale flat placement, *Proc. 34th ACM/IEEE Design Automation Conf.*, 1997, p. 746.

[22] Chu, C. and Viswanathan, N., FastPlace: efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model, *Proc. Int. Symp. on Physical Design*, 2004, p. 26.

[23] Hu, B. and Marek-Sadowska, M., FAR: fixed-points addition & relaxation based placement, *Proc. Int. Symp. on Physical Design*, 2002, p. 161.

[24] Hu, B., Zeng, Y., and Marek-Sadowska, M., mFAR: fixed-points-addition-based VLSI placement algorithm, *Proc. Int. Symp. on Physical Design*, 2005, p. 239.

[25] Eisenmann, H. and Johannes, F. M., Generic global placement and floorplanning, *Proc. 35th ACM/IEEE Design Automation Conf.*, 1998, p. 269.

[26] Breuer, M. A., Min-cut placement, *J. Design Automat. Fault Tolerant Comput.*, 1(4), 343, 1977.

[27] Dunlop, A. E. and Kernighan, B. W., A procedure for placement of standard-cell VLSI circuits, *IEEE Trans. CAD*, CAD-4(1), 92–98, 1985.

[28] Cong, J. and Smith, M., A parallel bottom-up clustering algorithm with applications to circuit partitioning in vlsi designs, *Proc. Design Automation Conf.*, 1993, p. 755.

[29] Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S., Multilevel hypergraph partitioning: application in VLSI Domain, *Proc. ACM/IEEE Design Automation Conf.*, 1997, p. 526.

[30] Caldwell, A. E., Kahng, A. B., and Markov, I. L., Improved algorithms for hypergraph partitioning, *Proc. Asia South Pacific Design Automation Conf.*, 2000.

[31] Fiduccia, C. M. and Mattheyses, R. M., A linear-time heuristic for improving network partitions, *Proc. Design Automation Conf.*, 1982, p. 175.

[32] Caldwell, A. E., Kahng, A. B., and Markov, I. L., Can recursive bisection produce routable placements? *Proc. 37th IEEE/ACM Design Automation Conf.*, 2000, p. 477.

[33] Roy, J. A., Papa, D. A., Adya, S. N., Chan, H. H., Ng, A. N., Lu, J. F., and Markov, I. L., Capo: robust and scalable open-source min-cut floorplacer, *Proc. Int. Symp. on Physical. Design*, 2005, p. 224.

[34] Yildiz, M. C. and Madden, P. H., Global objectives for standard cell placement, *Proc. Great-Lakes Symp. on VLSI*, 2001, p. 68.

[35] Agnihotri, A. R., Ono, S., and Madden, P., Recursive bisection placement: Feng shui 5.0 implementation details, *Proc. Int. Symp. on Physical Design*, 2005, p. 230.

[36] Yildiz, M. C. and Madden, P. H., Improved cut sequences for partitioning-based placement, *Proc. Design Automation Conf.*, 2001, p. 776.

[37] Caldwell, A. E., Kahng, A. B., and Markov, I. L., Optimal partitioners and end-case placers for standard-cell layout, *IEEE Trans. CAD*, 19(11), 1304, 2000.

[38] Villarrubia, P., Nusbaum, G., Masleid, R., and Patel, E. T., IBM RISC chip design methodology, *Proc. of ICCD*, 1989, p. 143.

[39] Kahng, A. B. and Reda, S., Placement feedback: a concept and method for better min-cut placements, *Proc. Design Automation Conf.*, 2004, p. 357.

[40] Wang, M., Yang, X., and Sarrafzadeh, M., Dragon2000: standard-cell placement tool for large circuits, *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, 2000, p. 260.

[41] Madden, P. H., Partitioning by iterative deletion, *Proc. Int. Symp. on Physical Design*, 1999, p. 83.

[42] Agnihotri, A. R., Yildiz, M. C., Khatkhate, A., Mathur, A., Ono, S., and Madden, P. H., Fractional cut: improved recursive bisection placement, *Proc. Int. Conf. on Computer-Aided Design*, 2003, p. 307.

[43] Hagen, L., Huang, J. H., and Kahng, A. B., On implementation choices for iterative improvement partitioning algorithms, *IEEE Trans. CAD,* 16(10), 1199, 1997.

[44] Caldwell, A. E., Kahng, A. B., and Markov, I. L., Iterative partitioning with varying node weights, *VLSI Design*, 11(3), 249–258, 2000.

[45] Adya, S. N., Chaturvedi, S., Roy, J. A., Papa, D. A., and Markov, I. L., Unification of partitioning, placement and floorplanning, *Proc. Int. Conf. on Computer-Aided Design*, 2004, p. 12.

[46] Cong, J., Romesis, M., and Shinnerl, J. R., Fast floorplanning by look-ahead enabled recursive bipartitioning, *IEEE Trans. CAD,* 25(9), 1719–1732, 2006.

[47] Cong, J., Romesis, M., and Shinnerl, J., Robust mixed-size placement under tight white-space constraints, *Proc. Int. Conf. on Computer-Aided Design*, 2005, p. 165.

[48] Adya, S. N. and Markov, I. L., Consistent placement of macro-blocks using floorplanning and standard-cell placement, *Proc. Int. Symp. on Physical Design*, 2002, p. 12.

[49] Cong, J., Romesis, M., and Shinnerl, J. R., fast floorplanning by look-ahead enabled recursive bipartitioning, *Proc. Asia South Pacific Design Automation Conf.*, 2005.

[50] Tsay, R. S., Kuh, E. S., and Hsu, C. P., Proud: A fast sea-of-gates placement algorithm, *IEEE Design Test Comput.*, 5(6), 44–56, 1988.

[51] Sigl, G., Doll, K., and Johannes, F. M., analytical placement: a linear or a quadratic objective function? *Proc. ACM/IEEE Design Automation Conf.*, 1991, p. 427.

[52] Brenner, U. and Rohe, A., An effective congestion-driven placement framework, *Proc. Int. Symp. on Physical Design*, 2002.

[53] Xiu, Z., Ma, J. D., Fowler, S. M., and Rutenbar, R. A., Large-scale placement by grid warping, *Proc. Design Automation Conf.*, 2004, p. 351.

[54] Vygen, J., Four-way Partitioning of Two-Dimensional Sets, Report 00900-OR, Research Institute for Discrete Mathematics, University of Bonn, Bonn, Germany, 2000.

[55] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., *Numerical Recipes in C: The Art of Scientific Comput.*, 2nd ed., Cambridge University Press, Cambridge, UK, 1993.

[56] Sankar, Y. and Rose, J., Trading quality for compile time: ultra-fast placement for FPGAs, *Symp. on FPGAs*, 1999, p. 157.

[57] Chan, T. F., Cong, J., Kong, T., and Shinnerl, J., Multilevel optimization for large-scale circuit placement, *Proc. Int. Conf. on Computer-Aided Design*, 2000, p. 171.

[58] Chan, T. F., Cong, J., Kong, T., Shinnerl, J., and Sze, K., An enhanced multilevel algorithm for circuit placement, *Proc. Int. Conf. on Computer-Aided Design*, 2003.

[59] Chan, T. F., Cong, J., Kong, T., and Shinnerl, J. R., Multilevel circuit placement, in *Multilevel Optimization in VLSICAD*, Cong, J. and Shinnerl, J. R., Eds., Kluwer Academic Publishers, Dordrecht, 2003.

[60] Kahng, A. B. and Wang, Q., Implementation and extensibility of an analytic placer, *IEEE Trans. CAD,* 24(5), 734, 2005.

[61] Kahng, A. B., Reda, S., and Wang, Q., Architecture and details of a high quality, large-scale analytical placer, *Proc. Int. Conf. on Computer-Aided Design*, 2005.

[62] Vorwerk, K. and Kennings, A. A., An improved multi-level framework for force-directed placement, *Proc. of DATE*, 2005, p. 902.

[63] Brandt, A. and Ron, D., Multigrid solvers and multilevel optimization Strategies, *Multilevel Optimization and VLSICAD*, chap. 1 Kluwer Academic Publishers, Dordrecht, 2002.

[64] Briggs, W. L., Henson, V. E., and McCormick, S. F., *A Multigrid Tutorial*, 2nd ed., SIAM, Philadelphia, 2000.

[65] Karypis, G., Multilevel hypergraph partitioning, in *Multilevel Optimization and VLSICAD*, Cong, J. and Shinnerl, J. R., Eds., Kluwer Academic Publishers, Dordrecht, 2002.

[66] Alpert, C., Kahng, A. B., Nam, G.-J., Reda, S., and Villarrubia, P., A semi-persistent clustering technique for vlsi circuit placement, *Proc. Int. Symp. on Physical Design*, 2005, p. 200.

[67] Cong, J. and Lim, S. K., Edge separability based circuit clustering with application to circuit partitioning, *Proc. Asia South Pacific Design Automation Conf.,* 2000, p. 429.

[68] Hu, B. and Marek-Sadowska, M., Wire length prediction based clustering and its application in placement, *Proc. Design Automation Conf.*, 2003.

[69] Hu, B. and Marek-Sadowska, M., Fine granularity clustering based placement, *IEEE Trans. CAD,* 23(4), 527–536, 2004.

[70] Nash, S. G. and Sofer, A., *Linear and Nonlinear Programming*, McGraw-Hill, New York, 1996.

[71] Gill, P. E., Murray, W., and Wright, M. H., *Practical Optimization*, Academic Press, London and New York, 1981.

[72] Evans, L. C., *Partial Diferential Equations*, American Mathematical Society, Providence, RI, 2002.

[73] Arrow, K., Huriwicz, L., and Uzawa, H., *Studies in Nonlinear Programming*, Stanford University Press, Stanford, CA, 1958.

[74] Morton, K. W. and Mayers, D. F., *Numerical Solution of Partial Differential Equations*, Cambridge University Press, Cambridge, UK, 1994.

[75] Fiacco, A. V. and McCormick, G. P., *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, Classics in Applied Mathematics, SIAM, Philadelphia, 1990.

[76] Dunlop, A. E., Agrawal, V. D., Deutsch, D. N., Jukl, M. F., Kozak, P., and Wiesel, M., Chip layout optimization using critical path weighting, *Proc. Design Automation Conf.*, 1984, p. 133.

[77] Nair, R., Berman, C. L., Hauge, P. S., and Yoffa, E. J., Generation of performance constraints for layout, *IEEE Trans. CAD,* 8(8), 860, 1989.

[78] Tsay, R. S. and Koehl, J., An analytic net weighting approach for performance optimization in circuit placement, *Proc. Design Automation Conf.*, 1991, p. 620.

[79] Kong, T., A novel net weighting algorithm for timing-driven placement, *Proc. Int. Conf. on Computer-Aided Design*, 2002, p. 172.

[80] Cheng, C.-L. E., RISA: accurate and efficient placement routability modeling, *Proc. Int. Conf. on Computer-Aided Design*, 1994, p. 690.

[81] Lou, J., Thakur, S., Krishnamoorthy, S., and Sheng, H., Estimating routing congestion using probabilistic analysis, *IEEE Trans. CAD,* 21(1), 32, 2002.

[82] Brenner, U. and Rohe, A., An effective congestion-driven placement framework, *IEEE Trans. CAD,* 22(4), 387, 2003.

[83] Mayrhofer, S. and Lauther, U., Congestion-driven placement using a new multi-partitioning heuristic, *Proc. Int. Conf. on Computer-Aided Design*, 1990, p. 332.

[84] Hagen, L. W., Huang, D. J.-H., and Kahng, A. B., Quantified suboptimality of VLSI layout heuristics, *Proc. Design Automation Conf.*, 1995, p. 216.

[85] Chang, C., Cong, J., Romesis, M., and Xie, M., Optimality and scalability study of existing placement algorithms, *IEEE Trans. CAD,* 23(4), 537–549, 2004.

[86] Ono, S. and Madden, P. H., On structure and suboptimality in placement, *Proc. Asia South Pacific Design Automation Conf.*, 2005.

[87] Kahng, A. B. and Reda, S., Evaluation of placer suboptimality via zero-change netlist transformations, *Proc. Int. Symp. on Physical Design*, 2005, p. 208.

[88] Wang, Q., Jariwala, D., and Lillis, J., A study of tighter lower bounds in LP relaxation based placement, *Proc. ACM Great Lakes Symp. on VLSI*, 2005, p. 498.

[89] Cong, J., Romesis, M., Shinnerl, J., Sze, K., and Xie, M., Locality and Utilization in Placement Suboptimality (Technical report), 2006.

[90] Granville, V., Krivanek, M., and Rasson, J. P., Simulated annealing: a proof of convergence, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6), 652–656, 1994.

[91] Ng, A. N., Aggarwal, R., Ramachandran, V., and Markov, I. L., solving hard instances of floorplacement, *ISPD*, 2006.

[92] Chan, T., Cong, J., and Sze, K., Multilevel generalized force-directed method for circuit placement, *ISPD '05: Proceedings of the 2005 International Symposium on Physical Design*, ACM Press, New York, NY, USA, 2005, pp. 185–192.

[93] Caldwell, A., Kahng, A. B., and Markov, I. L., Hierarchical whitespace allocation in top-down placement, *IEEE Trans. on CAD*, 22(11), 716–724, 2003.

[94] Alpert, C. J., Nam, G.-J., and Villarrubia, P., Effective free space management for cut-based placement, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 22(10), 1343–1353, 2003.

[95] Li, C., Xie, M., Koh, C., Cong, J., and Madden, P., Routability-driven placement and white space allocation, In *Proceedings of the International Conference on Computer-Aided Design,* 2004, pp. 394–401.

[96] Chan, T. F., Cong, J., Shinnerl, J. R., Sze, K., and Xie, M., Enhanced multilevel mixed-size placement, *ISPD '06: Proceedings of the 2006 International Symposium on Physical Design*, ACM Press, New York, NY, USA, 2006, pp. 212–214.

[97] Chan, T., Cong, J., and Sze, K., Multilevel generalized force-directed method for circuit placement, *ISPD '05: Proceedings of the 2005 International Symposium on Physical Design*, ACM Press, New York, NY, USA, 2005, pp. 185–192.

[98] Nam, G.-J., ISPD 2006 placement contest: benchmark suite and results, *ISPD '06: Proceedings of the 2006 International Symposium on Physical Design*, ACM Press, New York, NY, USA, 2006, p. 167.

[99] Nam, G.-J., and Cong, J., ed. *Modern circuit placement: best practices and results*, Springer, 2007, (to appear).

80

Multicommodity Flow Algorithms for Buffered Global Routing

Christoph Albrecht
Cadence Berkeley Labs

Andrew B. Kahng
University of California at San Diego

Ion Măndoiu
University of Connecticut

Alexander Zelikovsky
Georgia State University

80.1 Introduction

Due to delay scaling effects in deep-submicron technologies, interconnect planning and synthesis are becoming critical to meeting chip performance targets with reduced design turnaround time. In particular, the global routing phase of the design cycle is receiving renewed interest, as it must efficiently handle increasingly more complex constraints for increasingly larger designs (see Ref. [1] for a recent survey). In addition to handling traditional objectives such as congestion, wirelength, and timing, a critical requirement for next generations of global routers is the integration with other interconnect optimizations, most importantly with buffer insertion and sizing. Indeed, it is estimated that top-level on-chip interconnect will require up to 10^6 repeaters when we reach the 50 nm technology node. Since these repeaters are large and have a significant impact on global routing congestion, buffer insertion and sizing can no longer be done after global routing completes.

In this chapter, we present and enhance a powerful integrated approach introduced in Ref. [2] for congestion and timing-driven global routing, buffer insertion, pin assignment, and buffer/wire sizing. Our approach is based on a multicommodity flow formulation for the buffered global routing problem. Multicommodity flow-based global routing has been an active research area since the seminal work of Raghavan and Thomson [3]. Although the global routing problem is NP-hard (even highly restricted versions of it, see Ref. [4]), Raghavan and Thomson [3] have shown that the optimum solution can be approximated arbitrarily close in time polynomial in the number of nets and the inverse of the accuracy. To date, predictability of solution quality continues to be a distinct advantage of multicommodity flow-based methods over all other approaches to global routing, including popular rip-up-and-reroute approaches [1].

The original method of Raghavan and Thomson relies on randomized rounding of an *optimum* fractional multicommodity flow. Subsequent works [5,6] have improved runtime scalability by using the approximation algorithm for multicommodity flows by Ref. [7]. Yet, only the recent breakthrough improvements due to Garg and Könemann [8] and Fleischer [9] have rendered multicommodity flow-based global routing practical for full chip designs [10]. As Ref. [10], our algorithm is built upon the efficient multicommodity flow approximation scheme of Refs. [8,9]. Our main contribution is a provably good multicommodity flow-based algorithm that, for a given buffer site map, finds a buffered global routing minimizing buffer and wire congestion subject to given constraints on routing area (wirelength and number of buffers) and sink delays.

The key features of our approach include the following:

- Our implementation permits detailed floorplan evaluation in that it enables computing the tradeoff curve between routing area and wire/buffer congestion under any combination of delay and capacity constraints.
- Like the allocation heuristic in Ref. [11], our algorithm enforces maximum source/buffer wireloads and controls congestion by taking into account routing channel capacities and buffer site locations. At the same time, like the buffer-block planning algorithm in Ref. [12], our algorithm takes into account individual sink delay constraints.
- Simultaneously, our algorithm performs buffer and wire sizing by taking into account given libraries of buffer types and wire widths, and integrates layer and pin assignment (the latter with virtually no increase in runtime). Soft pin locations are modeled as multiple sites (grid locations), and are enabling to solution quality.

The rest of the chapter is organized as follows. In Section 80.2, we formalize the buffered global routing problem for 2-pin nets. Then, in Section 80.3.2, we reformulate the problem as a minimum cost integer multicommodity flow problem (with capacities on *sets of edges*), give an efficient algorithm for finding near-optimal solutions to the fractional relaxation, and show how to convert fractional solutions to near-optimal routings by randomized rounding. In Sections 80.4 and 80.5 we show how our approach can be extended to handle multipin nets as well as pin assignment, polarity constraints imposed by the use of inverting buffers, buffer and wire sizing, and prescribed delay upperbounds. We conclude the chapter with experimental results detailing the scalability and limitations of our algorithm and comparing it to the heuristic in Ref. [11].

80.2 Problem Formulation

In this section we formulate the buffered global routing problem. To simplify the presentation, we ignore pin assignment flexibility and assume that there is a single (noninverting) buffer type and a single wire width. We further assume that only buffer wireload constraints must be satisfied (i.e., ignore delay upper-bounds), and that each net has two pins only. Extensions of our approach to pin assignment, polarity constraints induced by the use of inverting buffers, buffer and wire sizing, timing constraints, and multipin nets are discussed in Section 80.5.

For a given floorplan and tile size, we construct a vertex- and edge-weighted *tile graph* $G = (V, E, b, w)$, $b : V \to \mathbb{N}$, $w : E \to \mathbb{N}$, where:

- V is the set of tiles;
- E contains an edge between any two adjacent tiles;
- For each tile $v \in V$, the *buffer capacity* $b(v)$ is the number of buffer sites located in v; and
- For each edge $e = (u, v) \in E$, the *wire capacity* $w(e)$ is the number of routing channels available between tiles u and v.

We denote by $\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$ the given netlist, where each net N_i is specified by a *source* s_i and a *sink* t_i.

A feasible solution to the buffered global routing problem seeks for each net N_i an $s_i - t_i$ path P_i buffered using the available buffer sites (see Figure 80.1) such that the source vertex and the buffers drive each at most U units of wire, where U is a given upper-bound (the example in Figure 80.1 has $U = 5$). Formally,

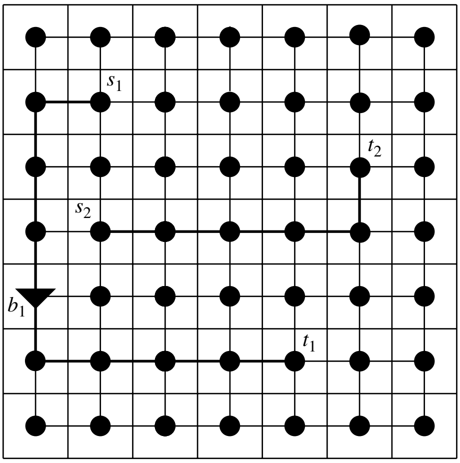


FIGURE 80.1 Tile graph with two 2-pin nets.

a *feasible buffered routing* for net N_i is a path $P_i = (v_0, v_1, \ldots, v_{l_i})$ in G together with a set of buffers $B_i \subseteq \{v_0, \ldots, v_{l_i}\}$ such that:

- $v_0 = s_i$ and $v_{l_i} = t_i$;
- $w(v_{i-1}, v_i) \geq 1$ for every $i = 1, \ldots, l_i$;
- $b(v_i) \geq 1$ for every $v_i \in B_i$; and
- The length along P_i between v_0 and the first buffer in B_i, between consecutive buffers, and between the last buffer and v_{l_i}, are all at most U.

We will denote by $\mathcal{R}_i$ the set of all feasible routings (P_i, B_i) for net N_i. Given buffered routings $(P_i, B_i) \in \mathcal{R}_i$ for each net N_i, the relative *buffer congestion* is

$$\mu = \max_{v \in V} \frac{|\{i : v \in B_i\}|}{b(v)}$$

and the relative *wire congestion* is

$$\nu = \max_{e \in E} \frac{|\{i : e \in P_i\}|}{w(e)}$$

The buffered paths (P_i, B_i), $i = 1, \ldots, k$, are simultaneously routable iff both $\mu \leq 1$ and $\nu \leq 1$. To leave resources available for subsequent optimization of critical nets and ECO routing, we will generally seek simultaneous buffered routings with buffer and wire congestion bounded away from 1. Using the total wire and buffer area as solution quality measure we get:

Integrated Global Routing and Bounded Wireload Buffer Insertion Problem[1]
Given:

- Grid-graph $G = (V, E, b, w)$, with buffer and wire capacities $b : V \rightarrow \mathbb{N}$, respectively $w : E \rightarrow \mathbb{N}$;
- Set $\mathcal{N} = \{N_1, \ldots, N_k\}$ of 2-pin nets with unassigned source and sink pins S_i, $T_i \subseteq V$; and
- Wireload, buffer congestion, and wire congestion upper-bounds $U > 0$, $\mu_0 \leq 1$, and $\nu_0 \leq 1$.

Find: Feasible buffered routings $(P_i, B_i) \in \mathcal{R}_i$ for each net N_i with relative buffer congestion $\mu \leq \mu_0$ and relative wire congestion $\nu \leq \nu_0$, minimizing the total wire and buffer area, that is, $\alpha \sum_{i=1}^{k} |B_i| + \beta \sum_{i=1}^{k} |P_i|$, where $\alpha, \beta \geq 0$ are given constants.

[1] The problem is called *Floorplan Evaluation Problem* in Ref. [2], but the formulation is useful in postplacement scenarios as well.

80.3 Buffered Global Routing via Multicommodity Flow Approximation

The high-level steps of our approach are the following:

1. Following Alpert et al. [11], we construct a two-dimensional tile graph to capture the number and spatial distribution of wire routing tracks and buffer insertion sites available in the given floorplan. For simplicity, we assume throughout the chapter that all tiles have the same size. As discussed in Ref. [13], uneven tiling, that is, using fine tiling in highly congested regions of the design and coarse tiling in regions blocked for routing or buffering, can be used to improve the trade-off between accuracy and solution time.
2. We then build an auxiliary graph in which every directed path from a net source to the net's sink captures a feasible wire route between them together with locations for the buffers to be inserted on this route such that buffer load constraints are satisfied. The auxiliary graph is obtained automatically from the tile graph using an original *gadget* construction (Section 80.3.1), which only increases the size of the graph by a linear factor.
3. We use the auxiliary graph to formulate the buffered global routing problem as an integer linear program (ILP). To formally express the ILP, we use a 0/1 variable for each source–sink path, and require that exactly one path be chosen for each source–sink pair. The objective is to minimize the wire and buffer congestion subject to a given upper-bound on the total wirelength (Section 80.3.1).
4. We find a near-optimal solution to the fractional relaxation of the above integer program. Although the integer program has exponential size (there are exponentially many variables corresponding to source-sink paths in the auxiliary graph), we give a combinatorial algorithm which runs in polynomial time by representing explicitly only nonzero variables (Section 80.3.2). The algorithm combines the general framework for multicommodity flow approximation introduced by [8,9] with some of the extensions described in Refs. [10,14].
5. Finally, we round the fractional solution to an integer solution using a heuristically enhanced version of the randomized rounding method originally proposed in Ref. [3] (Section 80.3.3).

In this section, we detail each step of our approach for the case of 2-pin nets, and also discuss efficient computation of the entire trade-off curve between routing area and congestion for a given floorplan (Section 80.3.4).

80.3.1 Gadget Graph and Integer Program Formulation

Recall that, for every feasible buffered routing in the tile graph $G = (V(G), E(G), b, w)$, the wireload of the source and of each buffer must be at most U. We start by defining an auxiliary directed graph H, which captures exactly these feasible buffered routings (see Figure 80.2). The graph H has $U + 1$ vertices $v^0, v^1, \ldots, v^U$ for each vertex $v \in V(G)$. The index of each copy corresponds to the *remaining wireload budget*, that is, the number of units of wire that can still be driven by the last inserted buffer (or by the net's source). Buffer insertions are represented in the gadget graph by directed arcs of the form (v^j, v^U)—following such an arc resets the remaining wireload budget up to the maximum value of U. Each undirected edge (u, v) in the tile graph gives rise to directed arcs (u^j, v^{j-1}) and (v^j, u^{j-1}), $j = 1, \ldots, U$, in the gadget graph. Note that the copy number decreases by 1 for each of these arcs, corresponding to a decrease of 1 unit in the remaining wireload budget. In addition, we add to H individual vertices to represent net sources and sinks. Each source vertex is connected by a directed arc to the Uth copy of the node representing the enclosing tile. Furthermore, *all* copies of the nodes representing enclosing tiles are connected by directed arcs into the respective sink vertices.

Formally, the graph H has vertex set

$$V(H) = \{s_i, t_i \mid 1 \leq i \leq k\} \cup \{v^j \mid v \in V(G), 0 \leq j \leq U\}$$

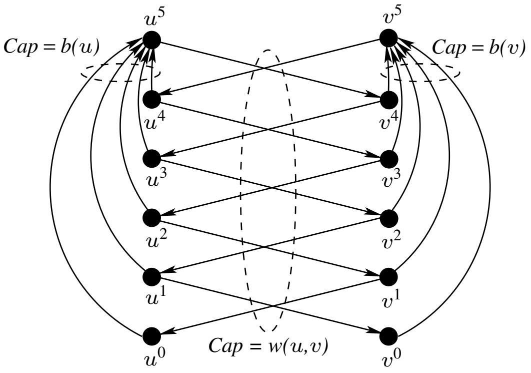


FIGURE 80.2 The basic gadget replacing edge (u, v) of the tile graph for buffer wireload upperbound $U = 5$.

and arc set

$$E(H) = E_{src} \cup E_{sink} \cup \Big(\bigcup_{(u,v) \in E(G)} E_{u,v} \Big) \cup \Big(\bigcup_{v \in V(G)} E_v \Big)$$

where

$$\begin{aligned} E_{src} &= \{(s_i, v^U) \mid \text{tile } v \text{ contains } s_i, 1 \le i \le k\} \\ E_{sink} &= \{(v^j, t_i) \mid \text{tile } v \text{ contains } t_i, 0 \le j \le U, 1 \le i \le k\} \\ E_{u,v} &= \{(u^j, v^{j-1}), (v^j, u^{j-1}) \mid 1 \le j \le U\} \\ E_v &= \{(v^j, v^U) \mid 0 \le j < U\} \end{aligned}$$

Each directed path in the gadget graph H corresponds to a buffered routing in the tile graph, obtained by ignoring copy indices for tile vertices and replacing each“buffer” arc (v^j, v^U) with a buffer inserted in tile v. Clearly, the construction ensures that the wireload of each buffer is at most U since a directed path in H can visit at most U vertices before following a buffer arc. Therefore, we get the following lemma.

Lemma 80.1

There is a one-to-one correspondence between the feasible buffered routings for net N_i in the tile graph G and the s_i−t_i paths in H.

We will use the correspondence established in Lemma 80.1 to give an ILP formulation for the buffered global routing problem. Let $\mathcal{P}_i$ denote the set of all simple s_i−t_i paths in H. We introduce a 0/1 variable x_p for every path $p \in \mathcal{P} := \cup_1^k \mathcal{P}_i$. The variable x_p is set to 1 if the buffered routing corresponding to $p \in P_i$ is used to connect net N_i, and to 0 otherwise. With this notation, the buffered global routing problem can be formulated as follows:

$$\min \sum_{p \in \mathcal{P}} \Big(\alpha \sum_{v \in V(G)} |p \cap E_v| + \beta \sum_{(u,v) \in E(G)} |p \cap E_{u,v}| \Big) x_p \tag{80.1}$$

subject to

$$\begin{aligned} &\sum_{p \in \mathcal{P}} |p \cap E_v|\, x_p \le \mu_0\, b(v) && v \in V(G) \\ &\sum_{p \in \mathcal{P}} |p \cap E_{u,v}|\, x_p \le \nu_0\, w(u, v) && (u, v) \in E(G) \\ &\sum_{p \in \mathcal{P}_i} x_p = 1 && i = 1, \ldots, k \\ &x_p \in \{0, 1\} && p \in \mathcal{P} \end{aligned}$$

ILP (80.1) is similar to the "path" formulation of the classical minimum cost integer multicommodity flow problem [15]. The only difference is that capacity constraints on the edges and vertices of the tile graph G become capacity constraints for sets of edges of the gadget graph H (see Figure 80.2). We note that the buffered global routing problem can be represented more compactly by using a polynomial number of edge-flow variables instead of the exponential number of path-flow variables x_p. However, we use formulation (80.1) since it is the natural setting for describing the approximation algorithm in the next section. The exponential number of variables is not impeding the efficiency of the approximation algorithm, which, during its execution, represents explicitly only a polynomial number of paths with nonzero flow.

80.3.2 The Approximation Algorithm

In this section, we give an efficient approximation algorithm that can be used for solving the fractional relaxation of ILP (80.1). Using an approach similar to that used in Ref. [8] for solving the minimum cost concurrent multicommodity flow problem (see also Ref. [10]), instead of solving the relaxation of ILP (80.1) directly we introduce an upper bound D on the wire and buffer area and consider the following linear program (LP):

$$\min \lambda \tag{80.2}$$

subject to

$$\sum_{p\in\mathcal{P}} \left(\alpha \sum_{v\in V(G)} |p\cap E_v| + \beta \sum_{(u,v)\in E(G)} |p \cap E_{u,v}|\right) x_p \le \lambda\ D$$

$$\begin{aligned}
&\sum_{p\in\mathcal{P}} |p\cap E_v|\, x_p \le \lambda\ \mu_0\, b(v) && v\in V(G)\\
&\sum_{p\in\mathcal{P}} |p\cap E_{u,v}|\, x_p \le \lambda\ \nu_0\, w(u,v) && (u,v)\in E(G)\\
&\sum_{p\in\mathcal{P}_i} x_p = 1 && i=1,\ldots,k\\
&x_p \ge 0 && p\in\mathcal{P}
\end{aligned}$$

Let λ^* be the optimum objective value for LP (80.2). Solving the fractional relaxation of ILP (80.1) is equivalent to finding the minimum D for which $\lambda^* \le 1$. This can be done by a binary search, which requires solving the LP (80.2) for each probed value of D. A lower bound on the optimal value of D can be derived by ignoring all buffer and wire capacity constraints, that is, by computing for each net N_i buffered paths $p \in \mathcal{P}_i$ minimizing $\alpha \sum_{v\in V(G)} |p\cap E_v| + \beta \sum_{(u,v)\in E(G)} |p\cap E_{u,v}|$. A trivial upper bound is the total routing area available, that is, $D_{\max} = \alpha\, \mu_0 \sum_{v\in V(G)} b(v) + \beta\, \nu_0 \sum_{(u,v)\in E(G)} w(u,v)$. In particular, unfeasibility in the fractional relaxation of ILP (80.1) is equivalent to λ^* being greater than 1 when $D = D_{\max}$, and can therefore be detected using the algorithm described below.

The algorithm for approximating the optimum solution to LP (80.2) (see Figure 80.3) uses the general framework for multicommodity flow approximation introduced in Ref. [8] combined with ideas similar to those in Ref. [14] for efficiently handling set capacity constraints. The algorithm relies on simultaneously approximating the *dual* LP:

$$\max \sum_{i=1}^{k} l_i \tag{80.3}$$

subject to

$$\sum_{v\in V(G)} \mu_0 b(v) y_v + \sum_{(u,v)\in E(G)} \nu_0 w(u,v) z_{u,v} + Du = 1$$

```
(1)  Set $y_v := \frac{\delta}{\mu_0 b(v)}\ \forall v \in V(G)$,   $z_e := \frac{\delta}{\nu_0 w(e)}\ \forall e \in E(G)$,   $u := \frac{\delta}{D}$
(2)  Set $x_p := 0\ \forall p \in \mathcal{P}$
(3)  Set $r = 0$ and $p_i := \emptyset$ for $i = 1, ..., k$.
(4)  While $\mu_0 \sum_{v \in V(G)} b(v) y_v + \nu_0 \sum_{(u,v) \in E(G)} w(u, v) z_{u,v} + Du < 1$ do:
(5)  begin
(6)  r := r+1.
(7)    For $i := 1$ to $k$, do
(8)    begin
(9)      If $p_i = \emptyset$ or $\sum_{v \in V(G)} |p_i \cap E_v|(y_v + \alpha u) + \sum_{(u,v) \in E(G)} |p_i \cap E_{u,v}|(z_{u,v} + \beta u) > (1 + \gamma\varepsilon) l_i$ then
(10)     begin
(11)       Find a path $p_i \in \mathcal{P}_i$ minimizing $l_i := \sum_{v \in V(G)} |p_i \cap E_v|(y_v + \alpha u) + \sum_{(u,v) \in E(G)} |p_i \cap E_{u,v}|(z_{u,v} + \beta u)$
(12)     end
(13)     Set $x_{p_i} := x_{p_i} + 1$
(14)     Set $y_v := y_v \left(1 + \varepsilon \frac{|p_i \cap E_v|}{\mu_0 b(v)}\right)\ \forall v \in V(G)$,   $z_e := z_e \left(1 + \varepsilon \frac{|p_i \cap E_{u,v}|}{\nu_0 w(u,v)}\right)\ \forall (u, v) \in E(G)$
           $u := u \left(1 + \varepsilon \frac{\alpha \sum_{v \in V(G)} |p_i \cap E_v| + \beta \sum_{(u,v) \in E(G)} |p_i \cap E_{u,v}|}{D}\right)$
(15)     end
(16)   end
(17)   Output $(x_p/r)_{p \in \mathcal{P}}$
```

FIGURE 80.3 Algorithm for approximately solving LP (80.2).

$$\begin{aligned} \sum_{v \in V(G)} |p \cap E_v|\,(y_v + \alpha u) + \sum_{(u,v) \in E(G)} |p \cap E_{u,v}|\,(z_{u,v} + \beta u) \geq l_i, \;\; p \in \mathcal{P}_i \\ y_v \geq 0 \quad v \in V(G) \\ z_e \geq 0 \quad e \in E(G) \end{aligned}$$

The algorithm starts with trivial solutions for LPs (80.2) and (80.3), and then updates these solutions over several phases. In each phase (lines 5–16 in Figure 80.3), one unit of flow is routed for each commodity; a feasible solution to LP (80.2) is obtained in line 17 after dividing all path flows by the number of phases. Commodities are routed along paths with minimum weight with respect to weights of $y_v + \alpha u$ for arcs in $E_v, v \in V(G)$, of $z_{u,v} + \beta u$ for arcs in $E_{u,v}, (u, v) \in E(G)$, and of 0 for all the other arcs (cf. LP [80.3]). The dual variables are increased by a multiplicative factor for all vertices/edges on a routed path; this ensures that dual weights increase exponentially with usage and thus often used edges are subsequently avoided [8].

Minimum-weight paths are computed in line 11 of the algorithm using Dijkstra's single-source shortest path algorithm. To reduce the number of shortest path computations, paths are recomputed only when their weight increases by a factor of more than $(1 + \gamma\epsilon)$ (see the test in line 9). This speed-up idea, first applied in Ref. [9] for the maximum multicommodity flow problem, has been shown in Ref. [10] to decrease the running time in practice while maintaining the same theoretical worst-case runtime. The proof of the following theorem can be found in Ref. [13].

Theorem 80.1

The algorithm in Figure 80.3 finds an $(1 + \epsilon_0)$*-approximation with* $O\left(\frac{1}{\epsilon_0^2 \lambda^*} k \log n\right)$ *minimum-weight path computations, using* $\epsilon = \min\left\{\frac{1}{\gamma}, \frac{1}{\gamma}(\sqrt{1 + \epsilon_0} - 1), \frac{1}{4}\left(1 - \left(\frac{1}{1+\epsilon_0}\right)^{1/6}\right)\right\}$ *and* $\delta = \left(\frac{1-\epsilon'}{n+m}\right)^{1/\epsilon}$*, where n and m are the number of vertices and edges of G, and* $\epsilon' := \epsilon(1 + \epsilon)(1 + \epsilon\gamma)$.

Remark 1. The dependence on λ^* in Theorem 80.1 can be eliminated by a scaling technique described in Ref. [9]. Thus, using a Fibonacci heap implementation of Dijkstra's algorithm to compute minimum-weight paths leads to a runtime of $O(\frac{1}{\epsilon_0^2} k(m + n \log n) \log n)$ for the algorithm in Figure 80.3.

Remark 2. Using ideas from Ref. [10], it can be shown that the algorithm in Figure 80.3 not only minimizes λ, but also "strives" for a lexicographically minimum solution with respect to the vector consisting of the relative buffer congestion of the vertices, the relative wire congestion of the edges, and the ratio between the total routing area and the upperbound D. This is particularly useful for the case when the floorplan is unroutable using the given buffer sites and wire tracks, since then the solution returned by the algorithm indicates where we should add more routing resources (by local perturbations of the floorplan) to reach routability. As noted above, this case is handled by running the algorithm with $D = D_{\max}$. If we want to completely ignore the constraint on total routing area (i.e., set $D = \infty$), the dual variable u is 0 throughout the whole execution of the algorithm and can thus be eliminated.

Remark 3. In a practical implementation, line 2 of the algorithm, which requires setting to zero an exponential number of variables, is not implemented explicitly. Rather, the algorithm keeps track only of the paths with nonzero flow, that is, those paths for which flow is augmented in line 13. Several alternatives for memory efficient representation of the nonzero flows are discussed in the next section.

80.3.3 Randomized Rounding

In the previous section we have presented an algorithm for computing near-optimal solutions to LP (80.2). The last step in solving the buffered global routing problem is to convert these fractional flows into feasible buffered routings for each net. We follow the randomized rounding technique proposed by Raghavan and Thomson [3], and route each net N_i by randomly choosing one of the paths $p \in \mathcal{P}_i$, where the probability of choosing path p is equal to the fractional flow x_p (recall that $\sum_{p \in \mathcal{P}_i} x_p = 1$, i.e., x_p is a probability distribution over $\mathcal{P}_i$). Since the fractional flows satisfy buffer and wire congestion constraints, it follows that (for large enough capacities) the relative congestion after rounding increases only by a small factor [3].

A direct implementation of the randomized rounding scheme requires storing explicitly all paths with nonzero flow. However, this is typically unfeasible due to the large memory requirement. An alternative implementation, originally suggested by Ref. [3], is to compute edge flows instead of path flows during the algorithm in Figure 80.3, and then implement randomized rounding by performing a random walk between the source and sink of each net. As noted in Ref. [14], performing the random walks *backward*, that is, from sinks toward sources, leads to reduced congestion for the case when a significant number of the 2-pin nets result from decomposition of multipin nets.

A simpler implementation, which requires storing a single path per net, is to interleave randomized rounding with the computation of the fractional flows x_p. In this implementation, we continuously update the path selected for each net, as follows. In first phase, the single path routed for each net becomes the net's choice with probability 1. In iteration $r > 1$, the path routed for net i replaces the previous selection of net i with a probability of $(r-1)/r$. It is easy to see that the path selected after t phases was selected by the net in phase $r = 1, \ldots, t$ with an equal probability of $1/r$, that is, the probability that a path p is the final selection is equal to the fractional flow x_p computed by the algorithm in Figure 80.3.

The results reported in Section 80.6 were obtained using yet another implementation. In our implementation we save the paths routed for each net in the *last* $K = 5$ *phases* of the algorithm in Figure 80.3 (note that the K paths resulting for each net need not be distinct). Then, we pick for each net one of the K saved paths, uniformly at random. To further improve the results, we repeat the random choices a large number of times (10,000 times in our implementation) and keep the choices that result in the smallest congestion or routing area (depending on the optimization criteria). We found this scheme, which has still reasonable memory requirements, to work better in practice than the other approaches (although, technically, it implements only a rough approximation of the probability distribution required by Ref. [3]).

80.3.4 Area and Congestion Trade-Off Curve

To evaluate a floorplan at an early stage of the design process, it is useful to find not only the minimum routing area needed for given bounds on μ_0 and ν_0 on the relative buffer and wire congestion, but also how the total routing area increases if we enforce a smaller congestion. Obviously, a floorplan is better if a smaller area increase is needed for the same decrease in congestion. Let us denote by $\Lambda(\mu, \nu)$ the minimum routing area needed for a fractional solution with relative buffer and wire congestion not more than μ and ν, respectively. In the following, we denote a fractional solution x_p, $p \in \mathcal{P}$ for LP (80.2), simply by a vector x. Let $A(x)$, $\mu(x)$, and $\nu(x)$ denote the total routing area, buffer congestion, respectively wire congestion of x. The proofs of the following lemmas can be found in Ref. [13].

Lemma 80.2

The function $(\mu, \nu) \longmapsto \Lambda(\mu, \nu)$ *is convex.*

Lemma 80.3

Let x *be an optimal solution of LP (80.2) for given* D, μ_0, *and* ν_0. *If there exists a solution* x' *with* $\max(\frac{\mu(x')}{\mu_0}, \frac{\nu(x')}{\nu_0}) < \max(\frac{\mu(x)}{\mu_0}, \frac{\nu(x)}{\nu_0})$, *then* $\Lambda(\mu(x), \nu(x)) = A(x)$

Lemma 80.3 shows that in certain cases we can derive a value $\Lambda(\mu, \nu)$ from an optimal solution of the LP (80.2), and thus the binary search suggested in Section 80.3.2 can be avoided. This suggests the following approach to computing the full area versus congestion trade-off curve. First, compute the feasible region (which is also convex) for μ and ν by ignoring the constraint on the area. Then solve the LP (80.2) for certain values of D, μ_0, and ν_0. If the solution is on the boundary of the feasible region, decrease D such that μ and ν increase, otherwise a new point for the area and congestion trade-off curve has been found.

80.4 Handling Multipin Nets

Although a majority of nets have only two pins, modern designs also contain an increasing number of multipin nets. To apply our approximation algorithm from the previous section, we need to reduce multipin nets to 2-pin nets or otherwise adjust the algorithm to handle multipin nets. In this section we consider several methods for decomposing multipin nets and present an extension of our algorithm to 3-pin nets; these alternatives provide a range of trade-offs between runtime and solution quality.

The standard reduction constructs the minimum spanning tree T over all k pins of a k pin net and then splits the net into $k - 1$ 2-pin nets each corresponding to a single edge in T. Although the wireload can be accurately taken in account, the inherent drawback of this approach is that for high fanout nets we may end up with unbalanced and overloaded buffering. Note that the star topology decomposition, in which the source is connected with each sink by a separate edge, suffers from the same drawback. Instead of spanning trees, we suggest to use buffered Steiner trees. The minimum buffered routing for the entire k-pin net can be found using one of the algorithms from Ref. [16]. Such routing has been shown to be very close to optimal and is convenient for handling high fanout nets—both sink and wire loads are taken into account. The resulting buffered routing tree T connects the vertices of degree 1 (corresponding to terminals), degree 2 (corresponding to buffers), and degrees 3 and 4 (corresponding to branching points for Steiner tree routing) (see Figure 80.4[a]). Our approach is to split T into smaller pieces (2- or 3-pin nets) and route them separately using the algorithm from the previous section. The resulting pieces have longer total wirelength, thus allowing more flexibility for congestion minimization. We distinguish three methods of splitting T.

Fixed branching and fixed buffering. The tree T is split into 2-vertex subgraphs by replacing each vertex v with the $deg(v)$ copies each corresponding to one of the incident edges (see Figure 80.4[b]). Each subgraph corresponds to a single edge of T and has a single source and a single sink. The number of resulting 2-pin nets is $k + p + b - 1$, where k is the number of terminals, p the number of Steiner points, and b the number of buffers in T. This decomposition is the least flexible—the positions of both buffers and Steiner points are fixed. Routing- and buffer-congestion minimization may require using very long detours with possible addition of extra buffers.

Fixed branching and flexible buffering. The tree T is split into 2-vertex subgraphs by replacing each branching vertex v with the $deg(v)$ copies, each corresponding to one of the incident edges (see Figure 80.4[c]).

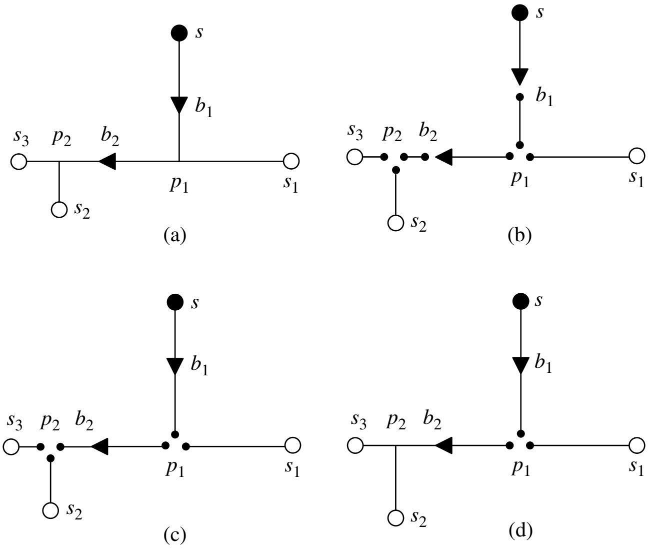


FIGURE 80.4 (a) Minimum buffered routing T of a 4-pin net with source s and sinks s_1, s_2, s_3 produced by the algorithm in Ref. [17]. T has two buffers b_1 and b_2 and two Steiner points p_1 and p_2. (b) Decomposition of T using fixed branching and fixed buffering. There are seven resulting 2-pin nets: (s, b_1), (b_1, p_1), (p_1, s_1), (p_1, b_2), (b_2, p_2), (p_2, s_2), and (p_3, s_3). (c) Decomposition of T using fixed branching and flexible buffering. There are five resulting 2-pin nets: (s, p_1), (p_1, s_1), (p_1, p_2), (p_2, s_2), and (p_3, s_3). (d) Decomposition of T using half-flexible branching and flexible buffering. The Steiner point p_1 is fixed and, therefore, split between three nets, while the Steiner point p_2 is flexible. There are two resulting 2-pin nets: (s, p_1), (p_1, s_1) and one 3-pin net (p_1, s_2, s_3).

Each subgraph corresponds to a single edge of the unbuffered version of T. The number of resulting 2-pin nets is $k + p - 1$. Similar to the previous decomposition, there is limited room for rerouting but now the buffers can be shifted between grid cells resulting in much better opportunities for buffer-congestion minimization. Since the buffer insertion may be caused by large sink load (e.g., in Figure 80.4[c], net (s, p_1) requires a buffer because of the three sinks downstream), it is necessary to compensate the upper bound U for some nets. This can be easily done by decreasing the level of the source of such net as follows. Normally, the source of a net is placed at the highest node of the edge gadget, for example, u^5 on Figure 80.2, but if compensation is necessary, then the source should be placed lower, for example, u^1, thus requiring a buffer much sooner.

Half-flexible branching and flexible buffering. Assume for simplicity that all Steiner points in T have degree 3, that is, no degree-4 Steiner points are allowed. The vertices of the unbuffered tree T can be colored into two, colors such that adjacent vertices have different color. If we fix locations of the Steiner points of the same color then all the resulting nets will have either three or two terminals (see Figure 80.4[d]). If we pick the color with the least amount of Steiner points, then we can be sure that at most half of all Steiner points are fixed implying that the number of resulting nets is at most $k - 1$. Indeed, at least $p/2$ Steiner points are not fixed and the corresponding three 2-pin nets for fixed branching are replaced with one 3-pin net. This reduces the total number of nets with respect to fixed branching by at least $2p/2 = p$.

This decomposition is the most flexible one, giving most opportunities for routing and buffer-congestion minimization. Unfortunately, it requires runtime-costly adjustments to the approximation algorithm from the previous section. Figure 80.5 gives the modified subroutine for computing feasible routings having minimum weight with respect to the dual variables. We assume here that the possible locations of the source pin for a net N_i are specified by S_i as before, while the two sinks are specified by sets T_i^1 and T_i^2. In the graph H we have vertices t_i^1 and t_i^2 and edge sets $\{(v^j, t_i^l) \mid v \in T_i^l, j = 0, \ldots, U\}, l = 1, 2$ for the sink pins of such a 3-pin net. For each possible Steiner point v, the algorithm tries all possible lengths j and k to the first buffer on the path from v to t_i^1, and respectively to t_i^2.

```
Set $w^* := \infty$
For all $v \in V$ do        // try all possible Steiner points
begin
    For $j := 0$ to $U$
    begin
        Find a shortest $v^{U-j} - t_i^1$–path $P_1$ in $H$
        For $k := 0$ to $U - j$
        begin
            Find a shortest $v^{U-k} - t_i^2$–path $P_2$ in $H$
            Find a shortest $s_i^0 - v^{U-j-k}$–path $P_0$ in $H$
            If $w(P_0) + w(P_1) + w(P_2) \leq w^*$ then
                Set $w^* := w(P_0) + w(P_1) + w(P_2)$; $T^* := P_0 \cup P_1 \cup P_2$
        end
    end
end
return $T^*$
```

FIGURE 80.5 Algorithm for finding minimum-weight buffered routings for 3-pin nets.

80.5 Extensions

In this section we describe how our approach to buffered global routing can be extended to handle pin assignment, polarity constraints imposed by the use of inverting buffers, buffer and wire sizing, and prescribed delay upperbounds. The modifications required to handle these extensions involve only changes to the gadget graph described in Section 80.3.1, but not to the approximation algorithm in Figure 80.3 or to the randomized rounding scheme.

80.5.1 Pin Assignment

At the early stages of the design flow there is a significant degree of flexibility available for pin assignment, and therefore the ability to exploit this flexibility can have a major impact on the quality of resulting global routings. Consideration of pin assignment requires only two small changes in the construction of the gadget graph described in Section 80.3.1: (1) source vertices s_i must now be connected by directed arcs to the Uth copies of *all* nodes representing enclosing tiles, and (2) copies $0, \ldots, U$ of *all* nodes representing enclosing tiles must be connected by directed arcs into the sink vertices t_i. Reading pin assignments from the paths selected by randomized rounding is trivial: we assign to each source (sink) an arbitrary pin in the tile visited first (last) by the selected path for the net.

The size of the gadget graph remains virtually unaffected by the pin assignment modification: for k nets we only add $O(k)$ edges to the gadget graph under the realistic assumption that each pin can be assigned to at most $O(1)$ tiles. Therefore, the time required to find minimum-weight paths, and hence the overall runtime of the algorithm in Figure 80.3, does not increase even though the number of paths available for each net increases when considering pin assignment.

80.5.2 Polarity Constraints

The basic problem formulation in Section 80.2 considers only a noninverting buffer type. In practice, inverting buffers are often preferred since they occupy a smaller area for the same driving strength. Although the use of inverting buffers introduces additional *polarity constraints*, which may require a larger number of buffers to be inserted, overall inverting buffers may lead to a better overall resource utilization. Algorithms for bounded capacitive load inverting (and noninverting) buffer insertion have been recently discussed in Ref. [16]. The focus of Ref. [16] is on single net buffering, with arbitrary positions for the buffers. In our approach, the goal is to minimize the overall number of buffers required by the nets, and buffers can be inserted only in the available sites.

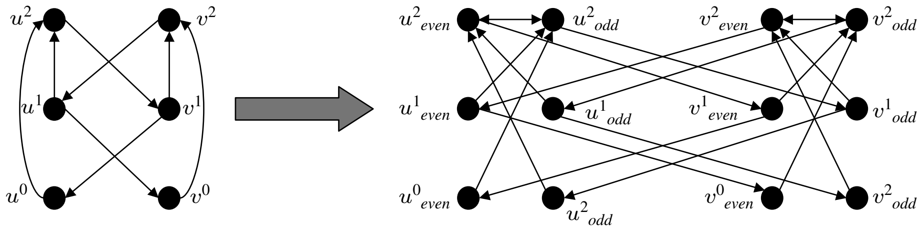


FIGURE 80.6 Gadget for polarity constraints with buffer load upperbound $U = 2$.

Consideration of polarity constraints required is achieved by modifying the basic gadget graph given in Section 80.3.1 as follows (see Figure 80.6). Each node of the basic gadget is replaced by an "even" and "odd" copy, that is, v^i is propagated into v^i_{even} and v^i_{odd}. Tile-to-tile arcs are replaced by two arcs connecting copies with the same polarity, for example, the arc (u^i, v^{i-1}) gives rise to $(u^i_{even}, v^{i-1}_{even})$ and $(u^i_{odd}, v^{i-1}_{odd})$. If a path uses such an arc, then it does not change polarity. Instead, each buffer arc changes polarity, that is, (v^i, v^U) gives rise to (v^i_{even}, v^U_{odd}) and (v^i_{odd}, v^U_{even}). The gadget also allows two inverting buffers to be inserted in the same tile for the purpose of meeting polarity constraints. This is achieved by providing bidirectional arcs connecting the Uth even and odd copies of a tile v, that is, (u^U_{even}, u^U_{odd}) and (v^U_{odd}, v^U_{even}). Finally, source vertices s_i are connected by directed arcs to the even Uth copy of enclosing tiles, and only copies of the desired polarity have arcs going into sink vertices t_i.

80.5.3 Buffer and Wire Sizing

Buffer and wire sizing are well-known techniques for timing optimization in the final stages of the design cycle [12]. However, early buffer and wire sizing can be equally effective for reducing congestion or wiring resources. In this section we show how to incorporate buffer and wire sizing in our algorithmic framework for global buffered routing. The key enablers to these extensions are again appropriate modifications of the gadget graph.

The gadget for buffer sizing is illustrated in Figure 80.7(a) for two available buffer sizes, one with wireload upperbound $U = 4$ and the other with wireload upperbound $U = 2$. The general construction entails using a number of copies of each tile vertex equal to the maximum buffer load upperbound U. For every buffer with wireload upperbound of $U' \leq U$, we insert buffer arcs $(v^i, v^{U'})$ for every $0 \leq i < U'$. Thus, the copy number of each vertex continues to capture the remaining wireload budget, which ensures the correctness of the construction.

Wire sizing (and a coarse form of layer assignment) can be handled by a different modification of the gadget graph (see Figure 80.7[b]). Assuming that per unit capacitances of the thinner wire widths are rounded to integer multiples of the "standard" per unit capacitance, the gadget models the use of thinner segments of wire by providing tile-to-tile arcs, which decrease the tile copy index (i.e., remaining wireload budget) by more than one unit. For example, in Figure 80.7(b), solid arcs (u^i, v^{i-1}) and (v^i, u^{i-1}) correspond to standard width connections between tiles u and v, while dashed arcs (u^i, v^{i-2}) and (v^i, u^{i-2}) correspond to "half-width" connections, that is, connections using wire with double capacitive load per unit.

While our models for buffer and wire sizing are rather coarse (e.g., we truncate all buffer wireload upperbounds to integer multiples of the tile dimension, ignore variations in input capacitances of buffers and sinks), we consider them to be sufficiently accurate first-approximations for driving these optimizations during the early physical design stages.

80.5.4 Delay Constraints

In Ref. [2] we have proposed a method for enforcing given sink delay constraints based on charging wiresegment delays to buffer arcs in the gadget graph, and using a routine for computing minimum-weight delay constrained paths instead of Dijkstra's algorithm in the algorithm for approximating the

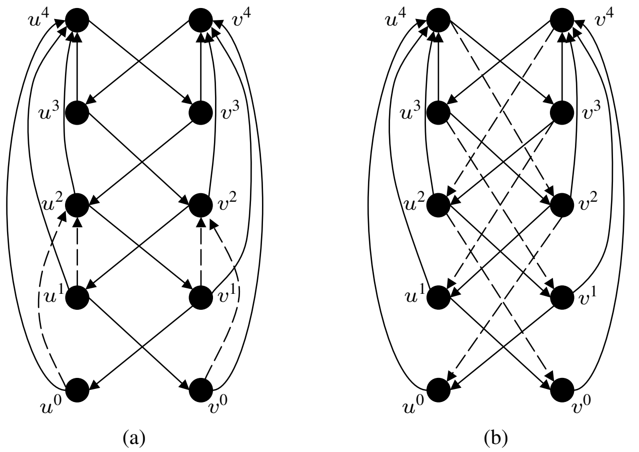


FIGURE 80.7 (a) Gadget for buffer sizing with two available buffer sizes, one with wireload upperbound $U = 4$ and the other with wireload upperbound $U = 2$. Solid arcs (u^i, u^4), respectively (v^i, v^4), correspond to the insertion of a buffer capable of driving 4 units of wire, while dashed arcs (u^i, u^2) and (v^i, v^2) correspond to the insertion of a smaller buffer capable of driving 2 units of wire. (b) Gadget for wire sizing with two available wire widths, standard width and "half" width (i.e., wire with double per unit capacitive load). The solid arcs (u^i, v^{i-1}) and (v^i, u^{i-1}) correspond to standard width connections between tiles u and v, while dashed arcs (u^i, v^{i-2}) and (v^i, u^{i-2}) correspond to half-width connections.

fractional solution to ILP (80.1). Here we give a different method for handling sink delay constraints. The new method is similar in spirit to the constructions in previous sections, relying exclusively on a modification of the gadget graph. In general, our construction applies for any delay model, such as the Elmore delay model, for which (1) the delay of a buffered path is the sum of the delays of the path segments separated by the buffers, and (2) the delay of each segment depends only on segment length and buffer parameters. However, for the sake of efficiency, segment delays would have to be rounded to relatively coarse units.

Figure 80.8 shows the gadget construction for the case when delay is measured simply by the number of inserted buffers. The idea is again to replicate the basic gadget construction, this time a number of times equal to the maximum allowed net delay. Within each replica, tile-to-tile arcs decrease remaining wireload budget by one unit. To keep track of path delays, buffer arcs advance over a number of gadget replicas equal to the delay of the wiresegment ended by the respective buffer (this delay can be easily determined for each buffer arc since the tail of the arc fully identifies the length of the wiresegment). The construction is completed by connecting net sources to the vertices with maximum remaining wireload budget in the "0 delay" replica of the gadget graph, and adding arcs into the sinks from all vertices in replicas corresponding to delays smaller than the given delay upperbounds.

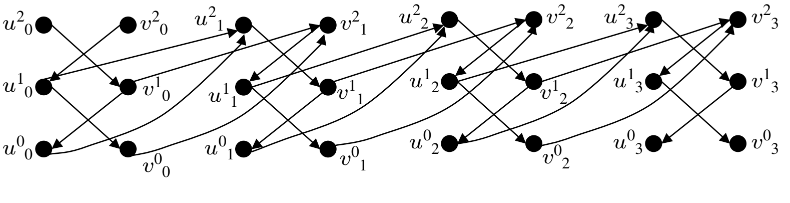


FIGURE 80.8 Gadget for enforcing delay constraints when the delay is measured by the number of buffers inserted between source and sink. The basic gadget is replicated a number of times equal to the maximum allowed net delay (3 in this example). Tile-to-tile arcs decrease remaining wireload budget within a gadget replica, while buffer arcs advance from one replica to the next.

Remark

An interesting feature of the resulting gadget graph is that it is acyclic. Hence, we can now compute minimum-weight paths in the approximation algorithm in Figure 80.3 by computing the distances from the source via a topological traversal of the graph in $O(m+n)$ time instead of the $O(m+n\log n)$ time needed by Dijkstra's algorithm.

80.6 Experimental Results

In this section we report results for a 2-pin net implementation of our multicommodity flow-based algorithm. All experiments with our algorithm were conducted on a 360 MHz SUN Ultra 60 workstation with 2 GB of memory, running under SunOS 5.7. The algorithm was coded in C and compiled using g++ 3.2 with -04 optimization. Unless otherwise noted, the value of precision parameter ϵ in the multicommodity flow algorithm was set to 0.3, and the number of iterations was limited to 64.

We tested the algorithm on the 10 circuits from Ref. [11], which are derived from testcases first used by Ref. [12]. We used the same circuit parameters as in the experiments for 2-pin nets of Ref. [11]; these parameters are summarized in Table 80.1. As in Refs. [11,12], we decomposed multipin nets into 2-pin nets by making direct connections from the source of a net to each of the net's sinks. Therefore, the numbers of 2-pin nets in Table 80.1 correspond to the numbers of sinks in Ref. [11]. We note that these numbers are slightly smaller than those reported in Ref. [12] since Ref. [11] retained only the nets requiring buffer insertion under the algorithm of Ref. [12]. We also note that the numbers of buffer sites used in our experiments are those used by Alpert et al. [11] in the experiments in which multipin nets were decomposed into 2-pin nets. These numbers were obtained directly from the authors, as they do not appear explicitly in Ref. [11] (the numbers of buffer sites given in Table 1 of Ref. [11] were used only in their experiments with undecomposed multipin nets; see Ref. [17] for more details on these experiments). Finally, we note that although we use the same grid sizes as Ref. [11], there are some small differences in tile areas between Table 80.1 and Ref. [11]. These differences, which are probably because of the different procedures used in rounding tile dimensions, are unlikely to affect to a measurable degree the results of the compared algorithms.

Table 80.2 shows the results of the multicommodity flow algorithm (with pin assignment) when run with $D=\infty$, that is, when the objective is to minimize the wire and buffer congestion only. The table shows that progressively better fractional solutions are obtained by the approximation algorithm. The results also show the trade-off between congestion on one hand and wiring resources (number of buffers and wirelength) on the other.

Table 80.3 gives the results for wirelength minimization (i.e., using $\alpha=0$ and $\beta=1$) subject to wire and buffer congestion constraints ($\mu_0=1.0$ and $\nu_0=1.0$). In these experiments the multicommodity flow

TABLE 80.1 Circuit Parameters

Circuit	# 2-Pin Nets	Grid Size	Tile Area	$w(e)$	Avg. Tiles per Pin	U	#Buffer Sites
a9c3	1526	30 × 30	1.09	52	4.9	6	32780
ac3	409	30 × 30	0.49	26	5.0	7	8550
ami33	324	33 × 30	0.46	32	5.0	6	17750
ami49	493	30 × 30	0.68	14	4.8	6	11450
apte	141	30 × 33	0.36	13	5.0	7	4200
hc7	1318	30 × 30	1.04	28	4.8	6	17780
hp	187	30 × 30	0.42	12	5.0	7	2350
Playout	1663	33 × 30	0.78	120	4.8	7	37550
xc5	2149	30 × 30	0.58	50	5.0	7	19150
Xerox	390	30 × 30	0.38	40	5.0	7	7000

Source: From Mandoiu, I. I., Recent advances in multicommodity flow algorithms for global routing, *Proc. 5th Int. Conf. on ASIC,* October 2003.

TABLE 80.2 Congestion Minimization Results ($D = \infty$) for the Multicommodity Flow Algorithm with $\varepsilon = 0.3$

		Wire	Buffer			
Circuit	Phase#	Congest	Congest	#Buffers	Wlen	CPU (s)
a9c3	1	0.75	0.80	3351	26057	12.0
	4	0.59	0.43	3356	26123	47.5
	16	0.51	0.23	3402	26595	188.8
	64	0.46	0.18	3505	27328	730.7
	64+ROUND	**0.62**	**0.30**	**3625**	**27980**	**785**
ac3	1	0.77	1.00	796	4998	3.0
	4	0.62	0.53	797	5008	12.2
	16	0.40	0.27	803	5072	48.9
	64	0.28	0.18	826	5211	192.3
	64+ROUND	**0.46**	**0.50**	**827**	**5251**	**213**
ami33	1	0.66	0.67	909	4466	2.6
	4	0.55	0.36	908	4476	10.6
	16	0.47	0.20	910	4515	42.2
	64	0.40	0.14	930	4618	163.0
	64+ROUND	**0.56**	**0.31**	**956**	**4698**	**181**
ami49	1	1.36	0.90	948	6045	3.0
	4	1.00	0.46	958	6083	12.3
	16	0.74	0.29	1040	6509	52.1
	64	0.66	0.21	1205	7278	211.3
	64+ROUND	**1.00**	**0.56**	**1308**	**7751**	**234**
apte	1	1.08	1.00	328	1668	1.2
	4	0.87	0.57	327	1677	5.0
	16	0.53	0.30	336	1725	21.1
	64	**0.44**	0.17	359	1836	86.8
	64+ROUND	**1.00**	**1.00**	**360**	**1841**	**98**
hc7	1	1.00	1.19	2203	17670	8.0
	4	0.79	0.61	2206	17738	32.5
	16	0.69	0.31	2301	18481	132.9
	64	0.62	0.23	2498	19660	516.9
	64+ROUND	**0.89**	**0.50**	**2675**	**20584**	**562**
hp	1	0.92	1.67	334	1952	1.3
	4	0.71	0.85	330	1961	5.2
	16	0.46	0.45	334	2003	21.8
	64	0.33	0.29	355	2119	89.8
	64+ROUND	**0.58**	**1.00**	**362**	**2147**	**101**
Playout	1	0.64	0.98	2890	23155	14.9
	4	0.52	0.42	2892	23199	60.6
	16	0.40	0.24	2922	23582	257.5
	64	0.33	0.17	3238	25809	1118.8
	64+ROUND	**0.36**	**0.28**	**3467**	**27281**	**1176**
xc5	1	1.14	1.31	3187	22314	17.6
	4	0.98	0.66	3202	22492	70.9
	16	0.74	0.37	3277	23231	288.2
	64	0.66	0.31	3570	24872	1134.7
	64+ROUND	**0.88**	**0.57**	**3895**	**26305**	**1216**
Xerox	1	0.93	1.42	659	3662	2.8
	4	0.72	0.77	660	3698	11.9
	16	0.45	0.40	684	3858	54.0
	64	0.32	0.21	753	4174	237.6
	64+ROUND	**0.47**	**0.67**	**779**	**4295**	**257**

TABLE 80.3 Wirelength Minimization ($\alpha = 0$ and $\beta = 1$) Subject to Wire and Buffer Congestion Constraints ($\mu_0 = 1.0$ and $\nu_0 = 1.0$)

Circuit	Algorithm	Wlen	%LB Gap	#Buffers	%LB Gap	Wire Congest	Buffer Congest	CPU (s)
a9c3	RABID	30723	5.64	4225	11.95	0.60	0.44	502
	MCF+ROUND	29082	0.00	3800	0.69	0.67	0.31	775
	MCF+PA+ROUND	26057	0.00	3378	0.81	0.62	0.30	779
ac3	RABID	5954	7.67	1037	15.74	0.58	0.33	208
	MCF+ROUND	5530	0.00	905	1.00	0.77	0.67	204
	MCF+PA+ROUND	4993	0.00	803	1.13	0.69	0.67	204
ami33	RABID	5232	6.93	1150	14.20	0.69	0.44	138
	MCF+ROUND	4893	0.00	1014	0.70	0.75	0.33	177
	MCF+PA+ROUND	4464	0.00	916	0.88	0.53	0.50	177
ami49	RABID	7592	11.87	1339	21.51	0.93	0.36	167
	MCF+ROUND	6792	0.07	1133	2.81	1.00	0.60	227
	MCF+PA+ROUND	6041	0.01	989	4.66	1.00	0.44	218
apte	RABID	2010	10.78	417	18.47	1.00	0.33	95
	MCF+ROUND	1833	1.03	377	7.10	1.00	1.00	88
	MCF+PA+ROUND	1663	0.15	331	4.75	1.00	1.00	87
hc7	RABID	21523	7.54	2983	17.44	0.82	0.35	386
	MCF+ROUND	20024	0.05	2591	2.01	0.96	0.47	551
	MCF+PA+ROUND	17660	0.00	2214	0.68	0.86	0.47	543
hp	RABID	2403	11.12	450	20.97	0.83	0.28	67
	MCF+ROUND	2165	0.13	404	8.60	1.00	1.00	95
	MCF+PA+ROUND	1945	0.00	345	6.81	0.92	1.00	94
Playout	RABID	27601	6.38	3840	15.04	0.45	0.64	813
	MCF+ROUND	25946	0.00	3429	2.73	0.53	0.34	1002
	MCF+PA+ROUND	23138	0.00	3011	4.37	0.40	0.32	995
xc5	RABID	27060	8.35	4410	23.25	0.84	0.81	694
	MCF+ROUND	25151	0.71	3843	7.41	0.98	0.62	1162
	MCF+PA+ROUND	22265	0.05	3341	4.90	1.00	0.65	1175
Xerox	RABID	4541	11.48	957	30.56	0.93	0.57	167
	MCF+ROUND	4078	0.12	805	9.82	1.00	1.00	212
	MCF+PA+ROUND	3658	0.00	692	6.30	0.88	0.67	208

Notes: RABID is the algorithm of Ref. [13], MCF the multicommodity flow algorithm without pin assignment capability, and MCF+PA the multicommodity flow algorithm with pin assignment enabled. Both MCF and MCF+PA were run with $\varepsilon = 0.3$.

algorithm is run once per testcase (without binary search), with D equal to the lower bound computed by routing each net optimally without taking into account capacity constraints. The multicommodity flow runtime includes randomized rounding (10,000 trials, as described in Section 80.3.3). RABID runtime is for an RS6000/595 workstation with 1 Gb of memory, as reported in Ref. [11].

The wirelength of the global routing obtained by our algorithm without pin assignment (MCF) is always within 1.03% of the lower bound. In contrast, the RABID heuristic of Ref. [11] exceeds the lower bound by 5.64–11.87%. To evaluate the effect of simultaneous pin assignment, we have added the possibility for each sink to be positioned not only in the given tile, but also in the 3–8 surrounding tiles (see Table 80.1 for the average number of tiles per pin of each testcase). Running our algorithm with pin assignment enabled (MCF+PA) further decreases wirelength by $\approx$ 10%, while being within at most 0.15% of the corresponding lower bound. We note that routing and pin assignment is performed by our algorithm in virtually the same time as routing alone.

Tables 80.4 gives results for the extension of the multicommodity flow algorithm to inverting buffer insertion, which is about twice slower than noninverting buffer insertion due to the doubling in size of the gadget graph. Inverter insertion leads to a very small increase in the number of buffers (because of the

TABLE 80.4 Wirelength Minimization Results for Noninverting versus Inverting Buffer Insertion

	Noninverting Buffers					Inverting Buffers				
Testcase	Wlen	#Buffers	W-congest	B-congest	CPU	Wlen	#Buffers	W-congest	B-congest	CPU
a9c3	29082	3800	0.67	0.31	775	29082	4540	0.60	0.41	1470
ac3	5530	905	0.77	0.67	204	5530	1095	0.69	0.50	417
ami33	4893	1014	0.75	0.33	177	4893	1186	0.62	0.29	359
ami49	6792	1133	1.00	0.60	227	6790	1417	1.00	1.00	449
apte	1833	377	1.00	1.00	88	1833	441	1.00	1.00	185
hc7	20024	2591	0.96	0.47	551	20024	3358	0.89	0.50	1030
hp	2165	404	1.00	1.00	95	2164	495	1.00	1.00	201
Playout	25946	3429	0.53	0.34	1002	25946	4235	0.53	0.32	1982
xc5	25151	3843	0.98	0.62	1162	25222	4799	0.96	1.00	2285
Xerox	4078	805	1.00	1.00	212	4155	1050	1.18	1.00	520

Notes: The number of buffer sites was assumed to be the same in both experiments.
Source: From Mandoiu, I. I., Recent advances in multicommodity flow algorithms for global routing, *Proc. 5th Int. Conf. on ASIC*, October 2003.

TABLE 80.5 Runtime Scaling for the Timing-Driven Version of the MCF Algorithm (Delay Measured by Number of Inserted Buffers)

	Delay Bound = 1		Delay Bound = 2		Delay Bound = 4		Delay Bound = 8		No Delay Bound	
Testcase	#Nets	CPU	#Nets	CPU	#Nets	CPU	#Nets	CPU	#Nets	CPU
a9c3	455	77	820	361	1361	2178	1526	7667	1526	775
ac3	152	29	249	122	374	666	409	2270	409	204
ami33	63	13	125	50	260	413	323	1761	324	177
ami49	177	25	298	113	442	598	493	2161	493	227
apte	49	12	67	33	126	255	141	968	141	88
hc7	569	70	873	305	1231	1584	1318	5217	1318	551
hp	76	15	124	63	174	330	187	1083	187	95
Playout	657	124	1095	651	1575	3506	1663	10979	1663	1002
xc5	1072	192	1429	748	2100	4158	2149	12351	2149	1162
Xerox	163	29	282	201	360	752	390	2308	390	212

Source: From Mandoiu, I. I., Recent advances in multicommodity flow algorithms for global routing, *Proc. 5th Int. Conf. on ASIC,* October 2003.

need to satisfy polarity constraints), which is easily compensated by the smaller size of inverters. At the same time, inverter insertion requires virtually the same wirelength and often gives improved congestion (except for the Xerox testcase).

Table 80.5 gives runtime scaling results for the extension of the multicommodity flow algorithm to buffered global routing with delay constraints. We note that the algorithm becomes faster for very tight delay constraints, since the number of nets that can meet delay constraints is only a fraction of the total number of nets. For moderately tight delay constraints almost all nets become routable, yet the runtime is comparable to that of the unconstrained version of the algorithm. For very lax delay constraints all nets become routable, and the runtime becomes significantly higher than that of the delay-oblivious version of the algorithm, by a factor roughly proportional to the increase in the size of the gadget graph, that is, the maximum delay upperbound. However, large delay constraints are not very useful since they are satisfied almost in totality by using the unconstrained version of the algorithm.

80.7 Conclusions

In this chapter we have presented the first provably good approach to buffered global routing with simultaneous timing- and congestion-driven buffered global route planning, pin and layer assignment, and wire/buffer sizing. The experimental results show that our method significantly outperforms approaches based on cascading individual optimizations such as the recent RABID algorithm of Alpert et al. [11]. Future work aims to incorporate in our implementation practical improvements such as the use of uneven-sized tiles, window constraints on buffer usage (as opposed to tile constraints), and faster-converging dual-update rules.

Acknowledgments

This work has been supported in part by Cadence Design Systems, Inc., the MARCO Gigascale Silicon Research Center, and NSF Grant CCR-9988331. Preliminary versions of these results have appeared in Refs. [2,18]. The authors wish to thank Charles Alpert, Jason Cong, and Jiang Hu for kindly providing them with the testcases used in Refs. [11,12].

References

[1] Hu, J. and Sapatnekar, S., A survey on multi-net global routing for integrated circuits, *Integration*, 31, 1, 2001.

[2] Albrecht, C., Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., Floorplan evaluation with timing-driven global wireplanning, pin assignment, and buffer/wire sizing, *Proc. Asia and South Pacific DAC and Int. Conf. on VLSI Design*, 2002, p. 580.

[3] Raghavan, P. and Thomson, C. D., Randomized rounding, *Combinatorica*, 7, 365, 1987.

[4] Vygen, J., Theory of VLSI Layout, Habilitation thesis, University of Bonn, 2001.

[5] Carden, R. C. and Cheng, C.-K., A global router using an efficient approximate multicommodity multiterminal flow algorithm, *Proc. DAC*, 1991, p. 316.

[6] Huang, J., Hong, X.-L., Cheng, C.-K., and Kuh, E. S., An efficient timing driven global routing algorithm, *Proc. DAC*, 1993, p. 596.

[7] Shahrokhi, F. and Matula, D. W., The maximum concurrent flow problem, *JACM*, 37(2), 318, 1990.

[8] Garg, N. and Könemann, J., Faster and simpler algorithms for multicommodity flow and other fractional packing problems, *Proc. FOCS,* 1998, p. 300.

[9] Fleischer, L. K., Approximating fractional multicommodity flow independent of the number of commodities, *SIAM J. Disc. Math.*, 13, 505, 2000.

[10] Albrecht, C., Global routing by new approximation algorithms for multicommodity flow, *IEEE Trans. CAD*, 20(5), 622, 2001.

[11] Alpert, C., Hu, J., Sapatnekar, S., and Villarrubia, P., A practical methodology for early buffer and wire resource allocation, *Proc. DAC*, 2001, p. 573.

[12] Cong, J., Kong, T., and Pan, D. Z., Buffer block planning for interconnect-driven floorplanning, *Proc. ICCAD*, 1999, p. 358.

[13] Albrecht, C., Kahng, A. B., Măndoiu, I. I., and Zelikovsky, A. Z., Multicommodity flow algorithms for buffered global routing, ACM Computing Res. Repository, cs.DS/0508045, 2005.

[14] Dragan, F. F., Kahng, A. B., Măndoiu, I. I., Muddu, S., and Zelikovsky, A. Z., Provably good global buffering by generalized multiterminal multicommodity flow approximation, *IEEE Trans. CAD*, 21, 263, 2002.

[15] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[16] Alpert, C., Kahng, A. B., Liu, B., Măndoiu, I. I., and Zelikovsky, A. Z., Minimum buffered routing with bounded capacitive load for slew rate and reliability control, *IEEE Trans. CAD*, 22, 241, 2003.

[17] Alpert, C., Hu, J., Sapatnekar, S., and Villarrubia, P., A practical methodology for early buffer and wire resource allocation, *IEEE Trans. CAD*, 22(5), 573, 2003.

[18] Măndoiu, I. I., Recent advances in multicommodity flow algorithms for global routing. *Proc. Conf. on ASIC*, 2003, p. 160.

81

Algorithmic Game Theory and Scheduling

Eric Angel
University of Evry Val d'Essonne

Evripidis Bampis
University of Evry Val d'Essonne

Fanny Pascual
University of Evry Val d'Essonne

81.1 Introduction

Nowadays, understanding the Internet is one of the major issues in theoretical computer science [1]. Toward this direction, a number of researchers consider the Internet as a system of rational selfish agents (or users), each acting his/her own profit. This simplification makes the use of many notions introduced in *noncooperative game theory* possible [2,3], where every agent determines her behavior (strategy) based on the other agents' strategies. The aim of every agent is to maximize her own individual profit, without taking into account the consequences of her choice to the other agents or to the system's performances. A central notion in this field is the notion of *Nash equilibrium* [2] defined as a combination of (deterministic or randomized) choices (strategies), one for each agent, from which no agent has an incentive to unilaterally change her strategy. The existence of such an equilibrium is assured for every finite game from the famous theorem of Nash [2]. However, even if, in the last few years, some progress has been made for particular classes of games [4,5], it remains an *open question* to decide whether or not the problem of computing Nash equilibrium is in general solvable in polynomial time. This question is actually among the most challenging open questions in theoretical computer science [1]. However, a much progress has been made in the related question of evaluating the *impact of the selfishness* of the agents on the efficient use of the system, or, in other words, on the *social welfare* measured in terms of an appropriate global objective function. Given that for a finite game there are, in general, a number of different Nash equilibria that may have different objective function values, this question is very similar to the one of evaluating the quality of a feasible solution for a combinatorial optimization problem. Exploiting this relation, Koutsoupias and Papadimitriou [6] adopted a *worst-case* approach and they introduced the notion of the *price of anarchy* (also known as *coordination ratio*), which is defined as the ratio of the value of the objective function in the *worst* Nash equilibrium and its value at the optimum. In fact, the price of anarchy evaluates the cost of the lack of *coordination*—as opposed to the *competitive ratio*, that evaluates the lack of *information* in the context of the *on-line* computation or the *approximation ratio*, that is used to evaluate the lack of *unbounded computational resources* in the context of the *off-line* computation.

In this chapter, we present some directions of research related to the notion of the price of anarchy and we illustrate them on three scheduling models involving *selfish agents*: for two of them, namely the KP (Koutsoupias–Papadimitriou) model [6] and the CKN (Christodoulou–Koutsoupias–Nanavati) model of Ref. [7], the agents are the *tasks*, while for the third one, the AT (Archer–Tardos) model [8], the agents are the *machines* (links of the network). Our aim is to offer a starting point for researchers who are interested in this area.

The chapter is organized as follows: in the remaining part of the introduction we present the three scheduling models that we use in the sequel. Section 81.2 is devoted to the notions of the price of anarchy and of the coordination mechanism that we illustrate for the KP and CKN models. In Section 81.3, we deal with the price of stability and the related topic of nashification. An example is given for the CKN model. Section 81.4 concerns the design of truthful algorithms. In the first part, we show how this can be done for the CKN model. In the second part, we give a general result characterizing the algorithms that do admit truthful payment schemes in the context of mechanism design and we illustrate it for the AT model. In Section 81.5, we give a brief state of the art and we point out some related references.

81.1.1 Selfish Tasks: The KP and CKN Models

In the KP and CKN models, n rational agents (tasks) $T_1, \ldots, T_n$ have to choose, among m available machines (also called links or processors) $P_1, \ldots, P_m$, the machine on which they will be scheduled. Each task T_i is characterized by a positive length (execution or processing time) l_i and by an identification number i, and each machine P_j is characterized by its identification number j. The strategy of each agent is either *pure*, that is, decides to be processed on a particular machine (with a probability one), or *mixed*, that is, the strategy is a probability distribution over the machines (goes on each machine with a given probability).

The choice of the machine on which the agent will be processed is based on the *cost* that is associated with each strategy. Let c_i^j be the cost of agent i if the agent chooses to be processed on machine j. In both models, the aim of each agent is to minimize her cost. Therefore, in a Nash equilibrium, the *cost* of agent i is

$$c_i = \min_j c_i^j$$

In the first model [6], called the **KP model**, the aim of each task T_i, $i = 1, \ldots, n$, is to minimize the *load* of the machine j on which she is executed, that is, the sum of the execution times of all the tasks that have been assigned to the same machine as T_i. If p_k^j denotes the probability of task T_k goes on machine j, then the (expected) cost of agent i on machine j is the expected load on machine j, that is,

$$c_i^j = l_i + \sum_{k \neq i} p_k^j l_k$$

In the second model [7], called the **CKN model**, every machine has a *public local policy* (known to all agents) that determines the order on which the tasks that are allocated to this machine will be scheduled. This policy may be a function of the lengths of the tasks, or of their identification numbers, or some other characteristics of the tasks (e.g., such a policy may consist in scheduling the tasks in decreasing order of their lengths). Tasks know the policies of the machines, and the aim of each task is to minimize her own completion time. The (expected) cost of agent i on machine j is then

$$c_i^j = l_i + \sum_{k \prec_j i} p_k^j l_k$$

where $k \prec_j i$ means that task T_k is scheduled before task T_i according to the policy of machine j.

In the sequel, we will consider for this latter model only pure strategies: the probability for a task to go on a given machine will be either 1 or 0.

81.1.2 Selfish Machines: The AT Model

In the AT model [8] there are m machines of different speeds and n tasks with lengths $l_1, \ldots, l_n$. The amount of time to complete task j on machine i is $\frac{l_j}{s_i}$, where s_i is the speed of machine i. Each machine is owned by an agent and the value of s_i is known only by agent i. Each agent has to report her speed, and based on the *reported* speeds, the mechanism $M = (A, \mathcal{P})$ constructs a schedule (i.e., an allocation of the tasks on the machines) using an algorithm A, and then pays the agents, according to a payment function $\mathcal{P}$, to compensate the cost induced by the processing of the tasks. Let $\mathcal{P}_i$ denote the amount of money (as computed by the payment function $\mathcal{P}$) given to agent i. The *profit* of agent i is defined as

$$\mathit{profit}_i = \mathcal{P}_i - \frac{w_i}{s_i}$$

that is, her payment minus the cost incurred by the agent in being assigned work w_i (w_i is the sum of the processing times of all the tasks that have been allocated to machine i). The *cost* for each agent is the opposite of her profit. The aim of each agent is to maximize her profit (i.e., to minimize her cost), and an agent may *lie* and bid a false value (a value different from her real speed) if this can increase her profit. The (social) cost of a schedule is the maximum load over all machines, that is, its *makespan.* The goal in this model is to design a mechanism (i.e., a schedule and a payment function), which is truthful (i.e., it assures that every agent has no incentive to lie), and which, at the same time, minimizes the makespan.

81.2 Price of Anarchy and Coordination Mechanisms

In many networks, users and service providers act selfishly, without an authority that monitors and regulates network operation to achieve some "social optimum." *How much performance is lost because of this?* This question, raised by Papadimitriou and Koutsoupias [6], opens many theoretical problems where the main question concerns the cost of the lack of *coordination*, called *price of anarchy.* This notion is studied in the next section and is illustrated in the case of the KP model. However, from an algorithmic point of view what is more interesting is to find ways for decreasing the price of anarchy in such a context. Such an approach has been introduced by Christodoulou et al. [7], where the notion of *coordination mechanism* is proposed. We discuss this notion in Section 81.2.2 and we illustrate it for the CKN model.

81.2.1 Price of Anarchy for the KP Model

In this section we consider the KP model and show that the price of anarchy on two identical machines is 3/2.

Let us consider an arbitrary task i and let us assume that she decides to go to machine j with a probability 1, that is, $p_i^j = 1$, then its cost, that is, the average load of machine j, is $c_i^j = l_i + \sum_{k \neq i} p_t^j l_k$. In the general case, the expected cost of task i is $\sum_j p_i^j c_i^j$. In a Nash equilibrium there is no incentive for task i to change its strategy, therefore it assigns nonzero probabilities only to machines j that minimize c_i^j. We note $c_i = \min_j c_i^j$.

For example, consider two tasks, each one with a length 1. The probabilities $p_i^j = 1/2$ for $i = 1, 2$ and $j = 1, 2$ give rise to a Nash equilibrium. Indeed, we have $c_1^1 = 1 + 1/2 \times 1 = 3/2$ and also $c_1^2 = c_2^1 = c_2^2 = 3/2$. What is the expected makespan of this Nash equilibrium? With a probability 1/4 (resp. 1/4) all tasks go to machine 1 (resp. 2) giving rise to a makespan of 2, with a probability 1/4 (resp. 1/4) task 1 goes to machine 1 (resp. 2) and task 2 goes to machine 2 (resp. 1), giving rise to a makespan of 1. The expected makespan is thus $1/4 \times 2 + 1/4 \times 2 + 1/4 \times 1 + 1/4 \times 1 = 3/2$. Note that the expected makespan is not the maximum over all machines of their average load. The average load of machine j is $M^j = \sum_i p_i^j l_i$, and it is 1 in this example. Of course, the optimum makespan of 1 is obtained when task 1 (resp. 2) goes to machine 1 (resp. 2). Recall that the price of anarchy is defined as the ratio between the (worst) expected makespan in a worst Nash equilibrium over the optimal makespan (achievable over the set of all schedules, i.e., not necessarily Nash equilibria). Therefore, this example shows that the price of anarchy on two identical machines is at least 3/2.

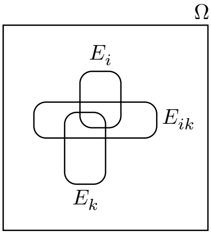


FIGURE 81.1 Let E_i (resp. E_k) denote the event "Task i (resp. k) is scheduled on the machine with the maximum load," and E_{ik} denote the event "Tasks i and k collide." One has $E_i \cap E_k \subseteq E_{ik}$. Hence, $q_i + q_k = \text{Proba}(E_i) + \text{Proba}(E_k) = \text{Proba}(E_i \cup E_k) + \text{Proba}(E_i \cap E_k) \leq 1 + \text{Proba}(E_{ik}) = 1 + t_{ik}$.

In the sequel we consider any Nash equilibrium, and we want to upper bound its expected makespan with respect to the optimum makespan, to show that this ratio 3/2 is tight.

The *contribution probability* q_i of task i is equal to the probability that this task is scheduled on the machine of maximum load (ties are broken using the machine with the smallest identification number). Therefore, the expected makespan is $\sum_i q_i l_i$.

Lemma 81.1 (Koutsoupias and Papadimitriou [6])
For any task i, one has $\sum_{k\neq i}(q_i + q_k)l_k \leq \frac{3}{2}\sum_{k\neq i} l_k$.

Proof
Let t_{ik} be the *collision probability* of tasks i and k, that is, the probability that they are scheduled on the same machine. Observe that if both tasks i and k contribute to the expected makespan, that is they are scheduled on the machine with the maximum load, then they must collide. This is illustrated in Figure 81.1.

Therefore, one has the following inequality:

$$q_i + q_k \leq 1 + t_{ik} \tag{81.1}$$

Moreover, we will show that for any task i

$$\sum_{k\neq i} t_{ik} l_k = c_i - l_i \tag{81.2}$$

We have $t_{ik} = \sum_j p_i^j p_k^j$, then $\sum_{k\neq i} t_{ik} l_k = \sum_j p_i^j \sum_{k\neq i} p_k^j l_k = \sum_j p_i^j (c_i^j - l_i) = \sum_j p_i^j (c_i - l_i)$. The last equality comes from the fact that since in a Nash equilibrium task i assigns nonzero probabilities only to machines j that minimize c_i^j, we have $p_i^j \neq 0$ when $c_i^j = c_i$ and $p_i^j = 0$ otherwise. Finally, since $\sum_j p_i^j = 1$ we obtain equality (81.2).

Now we will show that

$$c_i \leq \frac{\sum_k l_k}{2} + \frac{1}{2} l_i \tag{81.3}$$

Recall that $c_i^j = l_i + \sum_{t\neq i} p_t^j l_t = M^j + (1 - p_i^j) l_i$, with $M^j = \sum_i p_i^j l_i$ the expected load on machine j. Note that

$$M^1 + M^2 = \sum_k l_k \tag{81.4}$$

since $p_i^1 + p_i^2 = 1$ for all tasks i.

Therefore, we have

$$\begin{aligned} c_i &= \min_{j=1,2} c_i^j \\ &\leq \frac{1}{2}(c_i^1 + c_i^2) \end{aligned}$$

$$
\begin{aligned}
&\leq \frac{1}{2}\sum_{j=1}^{2}(M^j + (1 - p_i^j)l_i) \\
&= \frac{M^1 + M^2}{2} + \frac{1}{2}l_i \text{ since } p_i^1 + p_i^2 = 1 \\
&= \frac{\sum_k l_k}{2} + \frac{1}{2}l_i \quad \text{using Eq. (81.4)}
\end{aligned}
$$

and this proves Eq. (81.3).

Finally, we can write

$$
\begin{aligned}
\sum_{k\neq i}(q_i + q_k)l_k &\leq \sum_{k\neq i}(1 + t_{ik})l_k \quad \text{using (81.1)} \\
&\leq \sum_{k\neq i} l_k + \sum_{k\neq i} t_{ik}l_k \\
&= \sum_{k\neq i} l_k + c_i - l_i \quad \text{using (81.2)} \\
&\leq \sum_{k\neq i} l_k + \frac{\sum_k l_k}{2} - \frac{1}{2}l_i \quad \text{using (81.3)} \\
&= \frac{3}{2}\sum_{k\neq i} l_k
\end{aligned}
$$

□

Theorem 81.1 **(Koutsoupias and Papadimitriou [6])**

The price of anarchy for any number of tasks and two machines is 3/2.

Proof

We have already seen, with the example given in the beginning of this section, that the price of anarchy is at least 3/2. We show that is at most 3/2.

Let *OPT* be the optimum makespan. We will to compare the expected makespan in any Nash equilibrium versus the optimum makespan *OPT*.

Let us assume that $q_i \leq 3/4$ for all tasks i. Then the expected makespan is $\sum_k q_k l_k \leq \frac{3}{4}\sum_k l_k \leq \frac{3}{2}\,OPT$, since $OPT \geq \frac{1}{2}\sum_k l_k$.

Otherwise there exists a task i such that $q_i \geq 3/4$. In that case,

$$
\begin{aligned}
\sum_k q_k l_k &= \sum_{k\neq i} q_k l_k + q_i l_i \\
&\leq \frac{3}{2}\sum_{k\neq i} l_k - \sum_{k\neq i} q_i l_k + q_i l_i \quad \text{using Lemma 81.1} \\
&= \frac{3}{2}\sum_k l_k - \frac{3}{2}l_i - \sum_k q_i l_k + 2q_i l_i \\
&= \left(\frac{3}{2} - q_i\right)\sum_k l_k + \left(2q_i - \frac{3}{2}\right)l_i \\
&\leq \left(\frac{3}{2} - q_i\right)2\,OPT + \left(2q_i - \frac{3}{2}\right)OPT \quad \text{since } 2q_i - \tfrac{3}{2} \geq 0\text{, and using } OPT \\
&\geq \max\left\{\tfrac{1}{2}\textstyle\sum_k l_k, l_i\right\} \\
&= \frac{3}{2}\,OPT
\end{aligned}
$$

□

81.2.2 Coordination Mechanisms for the CKN Model

From an *algorithmic* point of view, an important question concerns possible ways to reduce the price of anarchy. One of the proposed approaches is due to Christodoulou et al. [7], who introduced the notion of *coordination mechanism*. A coordination mechanism is a set of local policies, one for each facility, that have to be based *only* on the characteristics of the local agents and not require any additional resources or alter the distributed nature of the system (the local policies are fixed once and for all before having any knowledge of the input).[1]

For instance, the local policy of a facility may give priorities to the agents or introduce delays. Knowing the coordination mechanism and the characteristics of the other agents, each agent chooses on which facility she goes, and she is then allocated to it, according to the policy of the facility.

The *price of anarchy* of a coordination mechanism is defined as the ratio between the objective function in the worst Nash equilibrium that we can obtain with this coordination mechanism, and the optimal value of the objective function (obtained in a centralized way, and which is not necessarily a Nash equilibrium).

Let us consider the following coordination mechanism for the CKN model on two machines: on each machine the tasks are sorted in order of increasing lengths: a task T_i is said larger than a task T_j if and only if the length of T_i is larger than the one of T_j $(l_i > l_j)$ or the tasks have the same lengths and the identification number of T_i is smaller than the one of T_j $(i < j)$. The first machine, denoted by P_{SPT}, schedules the tasks in order of increasing lengths, and the second machine, denoted by P_{LPT}, schedules the tasks in order of decreasing lengths.

With this mechanism, denoted by LPT-SPT, every task knows on which machine it will be scheduled first: a task T_i will go on P_{SPT} if the total length of the tasks that are smaller than T_i is smaller than or equal to the total length of the tasks that are larger than T_i; otherwise T_i will have incentive to go on P_{LPT}.

Theorem 81.2 **(Christodoulou et al.** [7]**)**

The coordination ratio of the LPT-SPT mechanism over two machines is 4/3.

Proof

Let us consider any schedule S obtained by this coordination mechanism, and let us show that the makespan of this schedule is smaller than or equal to $\frac{4}{3}\, OPT$, where OPT is the smallest makespan of any schedule involving the same tasks. Let T_r be the last task to be completed in S, and let C_{max} denote its completion time (i.e., C_{max} is the makespan of S).

Let L_1 (resp. L_2) be the sum of lengths of tasks scheduled on machine 1 (resp. 2), without taking into account task T_r. One has $C_{max} = \min\{L_1, L_2\} + l_r \leq (L_1 + L_2)/2 + l_r$. Therefore, $C_{max} \leq \frac{1}{2}\sum_{i \neq r} l_i + l_r \leq \frac{1}{2}\sum_{i=1}^{n} l_i + (1 - \frac{1}{2})l_r \leq OPT + \frac{1}{2}l_r$. If $l_r \leq \frac{2}{3}\, OPT$ then $C_{max} \leq \frac{4}{3}\, OPT$.

Let us now consider the case $l_r > \frac{2}{3}\, OPT$.

If T_r is scheduled on P_{LPT}, then the schedule is optimal. Indeed, if T_r is the only task of P_{LPT}, then S is an optimal schedule. In the other cases, there is at least one task before T_r on P_{LPT} and this task has a length larger than $\frac{2}{3}\, OPT$. Likewise, since task T_r has no incentive to go on P_{SPT}, there are on P_{SPT} tasks whose sum of lengths is larger than $\frac{2}{3}\, OPT$. Thus $\sum_{i=1}^{n} l_i > \frac{6\, OPT}{3} = 2\, OPT$, which is impossible.

If T_r is scheduled on P_{SPT}, then there exists a task T_1 of length larger than or equal to l_r on P_{LPT}. Let x be the sum of the lengths of the tasks scheduled before T_r on P_{SPT}. Since $l_1 + x + l_r \leq 2\, OPT$, and since $l_1 > \frac{2}{3}\, OPT$, we deduce that $l_r + x \leq \frac{4}{3}\, OPT$, that is, $C_{max} \leq \frac{4}{3}\, OPT$. □

We have shown that the coordination ratio of this mechanism is at most $\frac{4}{3}$. We can notice that this bound is tight by considering the following instance: two tasks of lengths 1 and two tasks of lengths 2. With this coordination mechanism the makespan will be 4, whereas the makespan of an optimal solution (where one task of length 1 and one task of length 2 are scheduled on each machine) is 3.

[1] This approach is related to the classical approach of game theory, the approach of *Mechanism Design*, where the agents are "paid" to cooperate [3].

However, this coordination mechanism cannot be generalized for m machines. Hence, Christodoulou et al. [7] proposed in the following coordination mechanism, that has a price of anarchy of $\frac{4}{3} - \frac{1}{3m}$: all the machines use a Longest Processing Time (LPT) local policy (i.e., every machine schedules its tasks in order of decreasing lengths) and introduce small (negligible) delays to break ties. In this way, it can be shown that the tasks have incentive to choose the machine on which they would have been affected by a centralized LPT algorithm (i.e., an algorithm which schedules greedily the tasks in the decreasing order of their lengths without leaving a machine idle whenever there is an unscheduled task). Thus, the price of anarchy of this mechanism is equal to the approximation ratio of the LPT algorithm (delays can be as small as we wish) [9]. In Ref. [7], the authors conjecture that there is no coordination mechanism with a better coordination ratio.

81.3 Price of Stability

In many applications, it is not true that the users (or agents) are interacting directly with each other. In reality, they interact with a protocol, which proposes a collective solution to all of them and the users are free to accept or reject. Hence, in these applications it is necessary to design protocols (algorithms) producing the *best* (or a near optimal) Nash equilibrium, that is, a *stable* (near) optimal solution such that no agent has incentive to defect from it [10]. A new measure for evaluating the impact of searching a solution under the constraint that the returned solution must be stable (i.e., a Nash equilibrium) has been introduced by Anshelevich et al. [10]: the *price of stability* is defined as the ratio of the objective function in the best Nash equilibrium and the global optimum (this maximum is taken over all instances). This measure can be viewed as the *optimistic* price of anarchy.

For the KP model there is always a pure Nash equilibrium with minimum makespan, so the price of stability is equal to 1 for this model. This result is a direct corollary of the results of Ref. [11] stating that for the KP model it is always possible to modify (*nashify*) an arbitrary initial solution to a pure Nash equilibrium without increasing the value of the makespan. However, for the CKN model this is not possible and the price of stability depends on the policies of the machines. Indeed, if for instance, we consider that the policies of the machines are LPT (each machine schedules the tasks in order of decreasing lengths), then the only pure Nash equilibrium is the schedule obtained by a centralized LPT schedule. Given that the approximation ratio of such an algorithm is $\frac{4}{3} - \frac{1}{3m}$ [9], the price of stability for the CKN model with LPT local policies is $\frac{4}{3} - \frac{1}{3m}$.

81.3.1 Approximate Stability for the CKN Model

In this section, we relax the definition of "stable" schedule: A solution is said to be *stable* if it corresponds to an *α-approximate Nash equilibrium*, that is, to a situation in which no agent has *sufficient incentive* to unilaterally change her behavior. We say that an agent does not have *sufficient incentive* to unilaterally leave the machine on which she is scheduled, if and only if this change does not increase her profit by more than α times its current profit, where α is a given threshold ($\alpha \geq 1$). If in a solution no agent has sufficient incentive to change strategy, then this solution is an α-approximate-Nash equilibrium. If $\alpha = 1$ then the schedule is an (exact) Nash equilibrium. Thus, we can define *the price of α-approximate stability* as the maximum ratio between the value of the objective function in the *best* α-approximate Nash equilibrium, and the value of the objective function in the global optimum.

Let us illustrate this notion for the CKN model with two identical machines whose policies are LPT. In the sequel, a task is said to be α-approximate if it will not reduce its completion time by a factor greater than α by changing machine.

We prove that this relaxation allows to reduce the price of stability to $(1 + \epsilon)$, while preserving an α-approximate Nash equilibrium, with α bounded by a constant. To do so, we consider the classical polynomial-time approximation scheme (PTAS) of Graham [9], modified slightly:

(1) Let k be some specified and fixed integer.

(2) Obtain an optimal schedule for the k largest tasks, such that:
  - Once tasks are assigned to each machine, they are scheduled on their machines in order of decreasing lengths (i.e., for a given machine, tasks are scheduled from the largest to the smallest one).
  - If two tasks have the same length, the one which has the smallest identification number is scheduled first.

(3) Schedule the remaining $n - k$ tasks using the LPT algorithm.

Note that since k is a constant, step (2) of this algorithm takes a polynomial time using an exhaustive enumeration search. This algorithm is a PTAS, and its approximation ratio is $1 + \varepsilon$, where ε is equal to $\frac{1}{2+2\lfloor \frac{k}{2} \rfloor}$, if the k largest tasks of the schedule are optimally scheduled [9]. Let us now show that this algorithm, denoted by OPT-LPT(k), always returns an α-approximate-Nash equilibrium, with $\alpha < k-2$, for the CKN model with two machines P_1 and P_2 whose policies are LPT.

Theorem 81.3 **(Angel et al. [12])**

The schedule returned by algorithm OPT-LPT(k) is an α-approximate-Nash equilibrium, with $\alpha < k - 2$, for two machines whose policies are LPT.

Proof

Let us show that each task of an OPT-LPT(k) schedule either does not have incentive to change machine (because she would increase its completion time by going on the other machine), or does not decrease her completion time by a factor larger than or equal to $k-2$, by going on the other machine. The $n-k$ smallest tasks of the schedule are scheduled using the LPT rule, so they do not have incentive to change machine. Thus we consider the k largest tasks. Let *OPT* be the optimal solution of these tasks, such as computed by OPT-LPT(k). We will now consider three cases. In the sequel we will denote the ith task of P_1 by x_i, the ith task of P_2 by y_i, and $l(t)$ will denote the length of task t. We will say that task t_1 is larger than task t_2 if $l(t_1) > l(t_2)$ or if $l(t_1) = l(t_2)$, and the identification number of t_1 is smaller than the identification number of t_2. Likewise t_1 is said smaller than t_2 if t_2 is larger than t_1.

- In the first case, there is, in *OPT*, only one task on a machine (w.l.o.g. on P_1), and $k - 1$ tasks on the other machine. Since this schedule is an optimal solution, the task on P_1 is necessarily the largest task on the schedule, and this schedule is an LPT schedule. So no task has incentive to change machine in this case.

- Let us now consider the case where there are exactly two tasks on a machine (w.l.o.g. on P_1) in *OPT*. The others $k - 2$ tasks are then on P_2.

We first show that no task scheduled on P_2 has incentive to go on P_1. By construction, we know that $l(x_1) + l(x_2)$ is larger than or equal to the sum of the lengths of the $k-3$ first tasks of P_2, $\sum_{j=1}^{k-3} y_j$. Let i be the largest number such that $l(x_1) \geq \sum_{j=1}^{i} y_j$: the $i+1$ first tasks of P_2 (i.e., the tasks who start at the latest at the end of x_1) do not have incentive to go on P_1, otherwise they would be scheduled after task x_1 and would not decrease their completion times. Moreover, we know that $l(x_2) \geq \sum_{j=i+2}^{k-3} y_j$: thus the tasks from y_{i+2} to y_{k-3} do not have incentive to change machine. Likewise, y_{k-2} does not have incentive to change: if it is smaller than x_2, then she would be scheduled on P_1 after x_2, and would not decrease her completion time, since *OPT* is an optimal solution. If y_{k-2} is larger than x_2, then y_{k-2} starts to be executed before (or at the same time as) x_2, otherwise by switching x_2 with y_{k-2} we could obtain a better solution than *OPT*. Thus, if it goes on P_1, y_{k-2} will be scheduled after x_1, and then will not decrease her completion time (note that y_{k-2} is smaller than x_1, otherwise *OPT* would not be an optimal solution).

The only task which may have incentive to change machine is x_2. If x_2 is smaller than all the other tasks, then she does not have incentive to change. Otherwise, since *OPT* is an optimal solution, we know that at least a task of P_2 starts at the same time or after x_2. In the best case, x_2 can go to the first position on P_2: by doing this, she starts on P_2 before at most $k-3$ tasks, which started before her when she was on P_1. These $k - 3$ tasks are smaller than x_2: the sum of their completion times, S, is thus smaller than $(k - 3)\, l(x_2)$.

The completion time of x_2 decreases with this change, from $S + l(x_2) < (k - 2)\, l(x_2)$ to $l(x_2)$. Thus x_2 is, in *OPT*, α-Nash-approximate, with $\alpha < k - 2$.

- Let us now consider the case where there are exactly $a < k - 2$ tasks on P_1, and $b < k - 2$ tasks on P_2. Let t be a task on P_1 (resp. P_2), who has incentive to change machine. When she changes machine, t overtakes p tasks of P_2 (resp. P_1), that is, she starts to be executed before p tasks, which started to be executed before t, that is before the change. We know that p is smaller than $k - 2$ because there are less than $k - 2$ tasks on each machine. Moreover, these tasks have a length smaller than the one of t, otherwise t would not overtake them. Thus, in the best case, t overtakes $k - 3$ tasks of length almost equal to $l(t)$, and the completion time of t decreases from a value smaller than $(k - 2)\, l(t)$ to $l(t)$. Thus t is α-Nash-approximate, with $\alpha < k - 2$. □

We saw that OPT-LPT(k) returns α-approximate Nash equilibria, with $\alpha < k - 2$. Let us now show that this bound is tight.

Theorem 81.4 **(Angel et al. [12])**

Let ε be any small number such that $0 < \varepsilon < 1$. OPT-LPT(k) can return α-approximate Nash equilibria, with $\alpha \geq k - 2 - \varepsilon$ and $k \geq 5$.

Proof

Let $\varepsilon' = \frac{\varepsilon}{k-2-\varepsilon}$, and let us consider the following instance: a task of length $k - 3 - \varepsilon'$, a task of length $1 + \varepsilon'$, and $k - 2$ tasks of length 1. The only optimal solution for this instance is the schedule where the tasks of length 1 are on the same machine (w.l.o.g. on P_2), and the two other tasks on the other machine. Let t denote the task of length $1 + \varepsilon'$: t is completed on P_1 at time $k - 2$. Note that $1 + \varepsilon' < k - 3 - \varepsilon'$ for any $0 < \varepsilon < 1$ and $k \geq 5$, thus by going on the other machine, t would end at time $1 + \varepsilon'$, and then decrease her completion time by $\frac{k-2}{1+\varepsilon'} = k - 2 - \varepsilon$. Thus t is $(k - 2 - \varepsilon)$-approximate. The schedule returned by OPT-LPT(k) on this instance is an α-approximate Nash equilibrium, with $\alpha \geq k - 2 - \varepsilon$. □

We can deduce from Theorem 81.3, and from the fact that the approximation ratio of OPT-LPT(k) is $\frac{1}{2+2\lfloor \frac{k}{2} \rfloor}$, the following result:

Corollary 81.1

Let k be any integer larger than or equal to 5. For the two machine scheduling game, if the local policies of the links are LPT, then the price of α-approximate stability is at most $1 + \varepsilon$, where $\varepsilon = \frac{1}{4+2\lfloor \frac{k}{2} \rfloor} < \frac{1}{k}$, for all $\alpha \geq k$.

81.4 Truthful Algorithms

Another important aspect of systems based on the selfish behavior of a set of agents is the notion of *truthfulness.* Consider a system with selfish agents, where the network is organized by a protocol whose aim is the maximization of the social welfare. The underlying assumption, in such a context, is that the agents on whom the protocols are applied are trustworthy. However, this assumption is unrealistic in some settings as the agents might try to manipulate the protocol by reporting false information to get some advantages. With false information, even the most efficient protocol may lead to unreasonable solutions if it is not designed to cope with the selfish behavior of the single entities. Thus, if every agent has a value (for a task, its length; for a machine, its speed; etc.), which is known only by herself, it is useful to design algorithms that are able to give incentive to the agents to bid their real secret values. Such algorithms are called *truthful.*

The field of *mechanism design* provides a theory to get truthful algorithms. The main idea of this theory is to pay the agents to convince them to declare their true values, thus helping the system to solve the optimization problem correctly. This is possible if every agent incurs some *monetary* cost. It is common to allow side payments to the agents, and to assume that each agent tries to maximize the sum of her payment and her intrinsic cost of the outcome. Thus a mechanism $M = (A, \mathcal{P})$ is a couple, where A is an algorithm that computes an allocation of the resources (in our case which affects tasks to machines), and $\mathcal{P}$ is a payment function that affects to every agent a certain amount of money. A mechanism is *truthful* if and only if the profit of every agent is maximized when the agent reveals her true value. The main general result in this direction is the Vickrey–Clarke–Groves (VCG) mechanism [13–15]. It handles arbitrary agents' cost functions and guarantees the truthfulness under the hypothesis that the global objective function is utilitarian (it optimizes the sum of the agents' costs or profits), and that the mechanism is able to compute the optimal solution. This is the case, for instance, when the agents are the tasks and they want to reduce their individual completion times, and the global objective function is the sum of completion times. However, if the objective function is the makespan, then the VCG mechanism cannot be used anymore since the minimization of the makespan is an NP-hard problem (and thus it is not possible to compute the optimum in polynomial time unless P = NP), and in addition the problem is not utilitarian since the goal is the minimization of the *maximum* completion time over all the machines.

In this section, we consider problems where the objective function is the makespan. In Section 81.4.1, we study the case where the agents are the tasks (CKN model): every agent is a task, which is the only one to know its real length and may lie on the value of this length. We present a (randomized) truthful mechanism without payment where the incentive of every agent is based only on her cost which is her (expected) completion time.

In Section 81.4.2, we consider the case where the agents are the machines (AT model): each agent is a machine that may lie on the value of its speed. We first show a fundamental result, due to Archer and Tardos [8], for the following type of problems: a set of loads have to be allocated by the mechanism on a set of machines (agents), and every agent has a secret data, which is a single positive real number that represents the cost incurred per unit of load assigned to this agent. These problems are known as *One-Parameter-Agent* problems. Archer and Tardos [8] showed that a mechanism $M = (A, \mathcal{P})$ for a problem with one-parameter agents is truthful if and only if algorithm A is *monotone*: an algorithm is monotone if, given the secret data $b_1, \ldots, b_n$ of the agents, then, for any i and fixed b_j ($j = 1, \ldots, n$ with $j \neq i$), the load assigned to agent i is nondecreasing with respect to b_i. Moreover, they show how to construct a payment function $\mathcal{P}_A$ such that if A is monotone then $(A, \mathcal{P}_A)$ is a truthful mechanism.

It is interesting to see that our selfish machines problem is a problem involving one-parameter agents: the agents are the machines, the loads are the tasks, and the agent's secret data is the inverse of her speed. Thus, an algorithm will be truthful for this problem if and only if it is monotone. Intuitively, monotonicity here means that increasing the speed of exactly one machine does not make the algorithm decrease the load assigned to that machine. We use this result in Section 81.4.2.2 to show that the LPT algorithm is truthful if the speeds of the machines are powers of a constant higher than or equal to 2.

81.4.1 Truthful Algorithm for Scheduling Selfish Tasks

We consider in this section selfish tasks whose aim is to reduce their completion times (CKN model). We assume that the length of each task (agent) is a private value, known only by herself, and that each agent bids a value representing her length. We focus on the following process: at first the agents declare their lengths; then given these bids the system allocates the tasks to the machines. The objective of the system is to minimize the makespan. The aim of each agent is to minimize her completion time and an agent may lie on this value if this can improve her (expected) completion time. We assume that the tasks cannot shrink their lengths and thus they will not bid values smaller than their real lengths, but they may bid values larger than their real lengths. There is a natural way to get a truthful deterministic algorithm, it is sufficient to schedule the tasks according to the *Shortest Processing Time* (SPT) algorithm, in which tasks are greedily scheduled from the smallest task to the largest task. The approximation ratio of this algorithm is $2 - \frac{1}{m}$ (see Ref. [9]). Here we will present a (randomized) truthful algorithm, which has an (expected) approximation ratio better than the one of SPT.

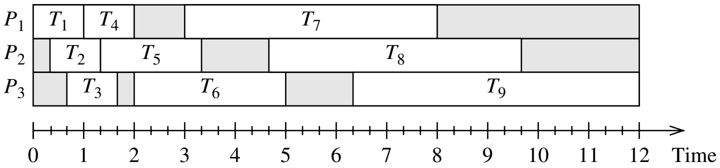


FIGURE 81.2 Example of an SPT$_\delta$ schedule.

Since SPT is truthful and LPT has a good approximation ratio, it is natural to wonder whether a randomized algorithm that returns the SPT schedule with a probability p and the LPT schedule with a probability $1-p$, is truthful for some values of p $(0 \le p \le 1)$. We can easily see that this algorithm is not truthful for any p smaller than 1. Indeed, consider the following instance on two machines: a task T_1 of length 1, a task T_2 of length 2, and a task T_3 of length 3. If T_1 bids its true value, then it will be scheduled first in the SPT schedule, and then finish at time 1; and it will be scheduled in the LPT schedule after task T_2, and then finish in this schedule at time 3. Thus, the expected completion time of T_1 if it bids its true value is $p + 3\,(1-p) = 3 - 2p$. If T_1 bids 2.5 (instead of its true value 1), then it will be on the first position in both the SPT and the LPT schedules. In both cases its completion time will be 1. Since the expected completion time of T_1 is smaller if it bids a false value rather than if it bids its true value, this algorithm is not truthful. We will now see that if we slightly modify the SPT algorithm, then a randomized algorithm of this type is truthful.

Let us consider the following algorithm, denoted by SPT$_\delta$ in the sequel:
Let $\{T_1, T_2, \ldots, T_n\}$ be n tasks to be scheduled on $m \ge 2$ identical machines, $\{P_1, P_2, \ldots, P_m\}$. Let us suppose that $l_1 \le l_2 \le \cdots \le l_n$. Tasks are scheduled alternatively on $P_1, P_2, \ldots, P_m$, in order of increasing lengths, and T_{i+1} starts to be executed when exactly $\frac{1}{m}$ of task T_i has been executed. Thus T_1 starts to be scheduled on P_1 at time 0, T_2 is scheduled on P_2 at time $\frac{l_1}{m}$, T_3 is scheduled on P_3 (on P_1 if $m = 2$) when $\frac{1}{m}$ of T_2 has been executed, that is, at time $\frac{l_1}{m} + \frac{l_2}{m}$, and so forth.

The schedule returned by SPT$_\delta$ will be called an SPT$_\delta$ schedule in the sequel. Figure 81.2 shows an SPT$_\delta$ schedule, where $m = 3$.

Let us now consider the following algorithm, denoted by LS_δ in the sequel:
Let m be the number of machines. With a probability of $\frac{m}{m+1}$, the output schedule is an SPT$_\delta$ schedule, and with a probability $\frac{1}{m+1}$, the output schedule is an LPT schedule.

We will now show that this algorithm has an approximation ratio better than the one of SPT (cf. Theorem 81.6) and that it is truthful (cf. Theorem 81.7). But first, we need to find the approximation ratio of SPT$_\delta$.

Theorem 81.5 (Angel et al. [16])

The algorithm SPT$_\delta$ is $(2 - \frac{1}{m})$-approximate: the makespan of an SPT$_\delta$ schedule is smaller than or equal to $(2-\frac{1}{m})$ OPT, where OPT is the makespan of an optimal schedule for the same tasks.

Proof

We have n tasks $T_1, \ldots, T_n$, such that $l_1 \le \cdots \le l_n$, to schedule on m machines. Each task T_i starts to be executed exactly when $\frac{1}{m}$ of T_{i-1} has been executed. So, if $n \le m$, then the makespan of the SPT$_\delta$ schedule is $\frac{1}{m}\,(l_1 + \cdots + l_{n-1}) + l_n \le \frac{1}{m}\,(n-1)\; l_n + l_n \le \frac{(2m-1)\,l_n}{m} \le (2-\frac{1}{m})\,l_n \le (2-\frac{1}{m})\,OPT$, since $l_n \le OPT$.

Let us now consider the case $n > m$. Let $i \in \{m+1, \ldots, n\}$. Task T_i starts to be executed when $\frac{1}{m}$ of T_{i-1} is executed, and T_{i-1} started to be executed when $\frac{1}{m}$ of T_{i-2} was executed, etc., $T_{(i-m)+1}$ started to be executed when $\frac{1}{m}$ of T_{i-m} was executed. So the idle time between T_i and T_{i-m} is $idle(i) = \frac{1}{m}\,(l_{i-m} + l_{i-m+1} + \cdots + l_{i-1}) - l_{i-m}$.

Let $i \in \{2, \ldots, m\}$. The idle time before T_i is equal to $idle(i) = \frac{1}{m}\,(l_1 + \cdots + l_{i-1})$, and there is no idle time before T_1, which starts to be executed at time 0. Thus, the sum of the idle times between tasks is $\sum_{i=2}^{n} idle(i) = \frac{1}{m}((m-1)l_{n-m+1} + (m-2)l_{n-m+2} + \cdots + l_{n-1})$.

Let $j \in \{n-m+1, \ldots, n-1\}$. Let $end(j)$ be the idle time in the schedule after the end of task T_j and before the end of T_n: $end(j) = l_{j+1} - \frac{m-1}{m} l_j + end(j+1)$, where $end(n) = 0$. So the sum of the idle times after the last tasks and before the end of the schedule is $\sum_{j=n-m+1}^{n-1} end(j) = (m-1)\left(l_n - \frac{m-1}{m} l_{n-1}\right) + (m-2)\left(l_{n-1} - \frac{m-1}{m} l_{n-2}\right) + \cdots + \left(l_{n-m+2} - \frac{m-1}{m} l_{n-m+1}\right)$.

The sum of the idle times on the machines, from the beginning of the schedule until the makespan, is the sum of the idle times between tasks (and before the first tasks), plus the sum of the idle times after the end of the last task of a machine and before the makespan. It is equal to $\sum_{i=2}^{n} idle(i) + \sum_{j=n-m+1}^{n-1} end(j) = \frac{1}{m}\left((m-1)l_{n-m+1} + (m-2)l_{n-m+2} + \cdots + l_{n-1}\right) + (m-1)\left(l_n - \frac{m-1}{m} l_{n-1}\right) + (m-2)\left(l_{n-1} - \frac{m-1}{m} l_{n-2}\right) + \cdots + \left(l_{n-m+2} - \frac{m-1}{m} l_{n-m+1}\right) = (m-1)\, l_n$.

Let ξ be the makespan of an SPT$_\delta$ schedule. ξ is the sum of the tasks plus the sum of the idle times, divided by m: $\xi = \frac{\left(\sum_{i=1}^{n} l_i\right) + (m-1)\, l_n}{m} = \frac{\sum_{i=1}^{n} l_i}{m} + \frac{(m-1)\, l_n}{m}$. Since $\frac{\sum_{i=1}^{n} l_i}{m} \leq OPT$ and $l_n \leq OPT$, we have $\xi \leq (2 - \frac{1}{m})\, OPT$. □

Theorem 81.6 (Angel et al. [16])

The expected approximation ratio of LS_δ *is* $2 - \frac{1}{m+1}\left(\frac{5}{3} + \frac{1}{3\,m}\right)$.

Proof

The approximation ratio of an SPT$_\delta$ schedule is $2 - \frac{1}{m}$ (see Theorem 81.5), and the approximation ratio of an LPT schedule is $\frac{4}{3} - \frac{1}{3\,m}$ (see Ref. [9]). Thus the expected approximation ratio of LS_δ is $\frac{m}{m+1}\left(2 - \frac{1}{m}\right) + \frac{1}{m+1}\left(\frac{4}{3} - \frac{1}{3\,m}\right) = \frac{1}{m+1}\left(2m - 1 + \frac{4}{3} - \frac{1}{3m}\right) = \frac{1}{m+1}\left(2(m+1) - \frac{5}{3} - \frac{1}{3m}\right) = 2 - \frac{1}{m+1}\left(\frac{5}{3} + \frac{1}{3\,m}\right)$. □

For example, in the case where we have two machines, the expected approximation ratio is $\frac{25}{18} < 1.39$, whereas the approximation ratio of SPT is $2 - \frac{1}{m} = 1.5$ in this case.

Let us now show that LS_δ is truthful.

Theorem 81.7 (Angel et al. [16])

The algorithm LS_δ *is truthful.*

Proof

Let us suppose that we have n tasks $T_1, \ldots, T_n$, ordered by increasing lengths, to schedule on m machines. Let us show that any task T_i does not have incentive to bid a length higher than her true length. Let us suppose that task T_i bids $b > l_i$, and that, by bidding b, T_i is now larger than all the tasks $T_1, \ldots, T_x$, and smaller than T_{x+1}. In the LPT schedule, the tasks T_{x+1} to T_n are scheduled in the same way, whatever T_i bids (l_i or b). By bidding b, T_i can, at best, start $(l_{i+1} + \cdots + l_x)$ time units before she had bided l_i. Thus the expected completion time of T_i in LS_δ decreases by at most $\frac{1}{m+1}\,(l_{i+1} + \cdots + l_x)$ time units when T_i bids b instead of l_i.

However, by bidding b instead of l_i, T_i will end later in the SPT$_\delta$ schedule: in this schedule, tasks from T_{i+1} to T_x will be started before T_i. Since a task T_j starts to be scheduled when $\frac{1}{m}$ of her predecessor T_{j-1} is executed, by bidding b, T_i starts $\frac{1}{m}\,(l_{i+1} + \cdots + l_x)$ time units later than she had bided l_i. Thus, the expected completion time of T_i in LS_δ is increased by $\frac{m}{m+1}(\frac{1}{m}\,(l_{i+1} + \cdots + l_x)) = \frac{1}{m+1}\,(l_{i+1} + \cdots + l_x)$. Thus, as a whole, the expected completion time of T_i cannot decrease when T_i bids a higher value than l_i, and we can deduce that LS_δ is truthful. □

81.4.2 Truthful Algorithms for Scheduling with Selfish Machines

We start this section by stating a general result, from Ref. [8], for one-parameter-agent problems whose application is illustrated in the case of the AT model.

81.4.2.1 Monotonicity

We characterize the algorithms that do and do not admit truthful payment schemes for mechanism design problems where the cost of agent i is of the form $t_i\ w_i(o)$, where t_i is her privately known cost per unit work and $w_i(o)$ is the amount of work—or load—assigned to her. This is, for instance, the case of scheduling problems where the agents are the machines that have to bid their speeds (see Section 81.4.2.2).

The private data t_i of agent i is only known by herself, whereas everything else is public knowledge. Every agent reports some value b_i to the mechanism: this value is called the agent's bid. Let b_{-i} denote the vector of bids, not including agent i. We write the vector of bids b as (b_{-i}, b_i). The mechanism's output algorithm computes a function $o(b)$ according to the agents' bids, and tries to optimize a global objective function without knowing t directly. Each agent i incurs some monetary cost, $cost_i(t_i, o) = t_i\, w_i(o)$. To offset these costs, the mechanism makes a payment $\mathcal{P}_i(b)$ to agent i. We assume that agent i always attempts to maximize her profit: $profit_i(t_i, b) = \mathcal{P}_i(b) - cost_i(t_i, o(b))$.

Theorem 81.8 (Archer and Tardos [8])

The output function $o(b)$ admits a truthful payment scheme if and only if it is decreasing. In this case, the mechanism is truthful if and only if the payments $\mathcal{P}_i(b_{-i}, b_i)$ are of the form

$$h_i(b_{-i}) + b_i\, w_i(b_{-i}, b_i) - \int_0^{b_i} w_i(b_{-i}, u)\, du \qquad (81.5)$$

where the h_i's are arbitrary functions.

Proof

Let us show this theorem with the pictorial proof of Figure 81.3.

In Figure 81.3 *(left)*, A, B, C, and D denote the areas of the rectangles they label. The cost of agent i is her privately known cost per unit work (t_i), times the amount of work assigned ($w_i(b_{-i}, b_i)$). If i's true value is y, her cost will be $B + D$ if she bids x, and $A + B + C + D$ if she bids y. Thus she would save cost $A + C$ by bidding x. If her true value is x, her cost will be B if she bids x, and $A + B$ if she bids y: she would incur an extra cost of A by bidding y. To motivate truth-telling, the extra payment for bidding y instead of x should then be at least $A + C$ and at most A, which is impossible since $C > 0$. Therefore, the work curve must decrease.

In Figure 81.3 *(right)*, the work curve is decreasing and the payments are given by Eq. (81.5). Geometrically, the payment to i if she bids x is a constant *cste* minus the area between the work curve and the horizontal line at height $w_i(x)$. If agent i's true value is t_i and she bids $x > t_i$, then her cost decreases by A (this cost is $A + C$ if the agent bids t_i whereas it is only C if she bids x), but her payment decreases by $A + B$ (this payment is *cste* $- D$ if the agent bids t_i whereas it is *cste* $- (D + A + B)$ if she bids x). Since $B > 0$, the agent never benefits from overbidding. Similarly, we can show that she never benefits from underbidding. □

81.4.2.2 A Truthful Algorithm for the AT Model

We now consider the AT model in which each machine is an agent whose secret data is her speed. We consider the case where we have two machines, and we wish to know if the LPT algorithm is truthful. The

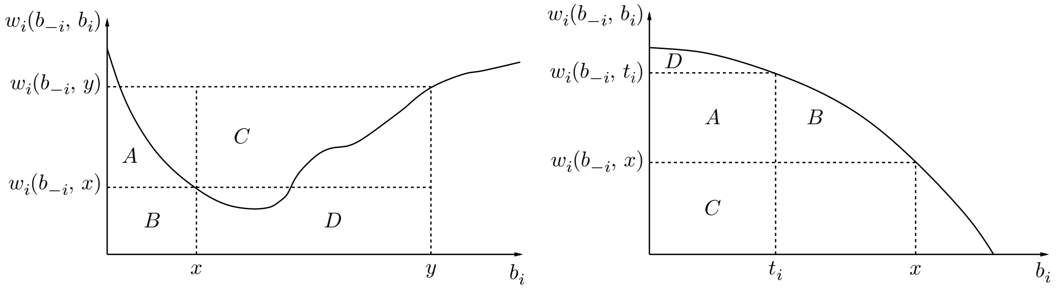


FIGURE 81.3 *Left:* The graph shows why the work curve must be decreasing. *Right:* The graph shows why agent i never gains by overbidding.

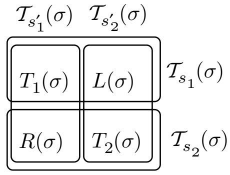

$\mathcal{T}_{s'_1}(\sigma)$
$\mathcal{T}_{s'_2}(\sigma)$
$T_1(\sigma)$
$L(\sigma)$
$\mathcal{T}_{s_1}(\sigma)$
$R(\sigma)$
$T_2(\sigma)$
$\mathcal{T}_{s_2}(\sigma)$

FIGURE 81.4 The set of tasks σ is partitioned into two subsets $\mathcal{T}_{s_1}(\sigma)$ and $\mathcal{T}_{s_2}(\sigma)$ (resp. $\mathcal{T}_{s'_1}(\sigma)$ and $\mathcal{T}_{s'_2}(\sigma)$) according the machine (1 or 2) on which each task is scheduled when the speed vector is s (resp. s').

LPT algorithm, in the case where the machines do not necessarily have the same speeds, is the following one: tasks are scheduled in order of decreasing lengths and each task is scheduled on the machine on which it will be completed first. Theorem 81.9 shows that LPT is truthful if the speeds of the machines are c-divisible, with $c \geq 2$, that is, if the speeds of the machines are restricted to be powers of c, where c is a constant greater than or equal to 2. It has also be shown in Ref. [17] that LPT is not truthful in the case of two machines with c-divisible speeds if $c \leq 1.78$. To prove Theorem 81.9, we will show that LPT is monotone when the speeds are c-divisible, with $c \geq 2$. By Theorem 81.8, shown in the previous section, this result implies that LPT is truthful in that case. Let us first start with the following lemma:

Lemma 81.2 (Ambrosio and Auletta [17])

For each speed vector s, where s_i is the speed of machine i, and for each sequence of tasks σ, the schedule computed by any list scheduling algorithm[2] on input s and σ is such that for any i, j, if $s_j \geq 2\, s_i$ then $w_j \geq w_i$, where w_i is the load assigned to machine i by the algorithm.

Proof

Suppose, by contradiction, that $w_j < w_i$ and consider the last task t assigned to machine i. We have $\frac{w_j+t}{s_j} < \frac{2\,w_i}{s_j} \leq \frac{w_i}{s_i}$, which contradicts the hypothesis that the list scheduling algorithm assigned task t to machine i. □

Consider two speed vectors $s = \langle s_1, s_2 \rangle$ and $s' = \langle s'_1, s'_2 \rangle$, where s' differs from s only on the speed of machine i and $s_i \leq s'_i$.

Let $\mathcal{T}_{s_1}(\sigma)$ be the set of tasks of σ that are assigned to machine 1 when the speed vector is s. We define $\mathcal{T}_{s_2}(\sigma)$, $\mathcal{T}_{s'_1}(\sigma)$ and $\mathcal{T}_{s'_2}(\sigma)$ in a similar way. We define $L(\sigma) = |\mathcal{T}_{s_1}(\sigma) \cap \mathcal{T}_{s'_2}(\sigma)|$, $R(\sigma) = |\mathcal{T}_{s_2}(\sigma) \cap \mathcal{T}_{s'_1}(\sigma)|$, $T_1(\sigma) = |\mathcal{T}_{s_1}(\sigma) \cap \mathcal{T}_{s'_1}(\sigma)|$ and $T_2(\sigma) = |\mathcal{T}_{s_2}(\sigma) \cap \mathcal{T}_{s'_2}(\sigma)|$. In other words, $L(\sigma)$ is, for example, the sum of the lengths of the tasks of σ that are assigned to machine 1 with respect to s (i.e., when the speed vector is s), and to machine 2 with respect to s' (i.e., when the speed vector is s'). These notations are depicted in Figure 81.4.

In the sequel we omit the argument σ when it is clear from the context.

Theorem 81.9 (Ambrosio and Auletta [17])

For any $c \geq 2$, the algorithm LPT is monotone when restricted to the case of two machines with c-divisible speeds.

Proof

Suppose, by contradiction, that LPT is not monotone for c-divisible speeds. Then, there exists two speed vectors $s = \langle s_1, s_2 \rangle$, and $s' = \langle s'_1, s'_2 \rangle$, where s' has been obtained from s by increasing only one speed (i.e., $s'_i \geq c\, s_i$ and $s'_j = s_j$ for i, $j \in \{1, 2\}$ and $i \neq j$), and a sequence of tasks $\sigma = \langle \sigma', t \rangle$ such that the scheduling of the tasks in σ computed by LPT with respect to s and s' is not monotone. Without

[2] A *list scheduling algorithm* is an algorithm, which schedules the tasks following the order of an arbitrary priority list without leaving any unnecessary idle time.

loss of generality, assume that σ is the shortest sequence that LPT schedules in a not monotone way: the schedule of σ' is monotone while the allocation of t destroys the monotonicity. We distinguish three cases.

- Let us first consider the case where $s'_1 = s_1$ and $s'_2 \geq c\, s_2$. Since the schedule of σ' is monotone, we know that $w_2(\sigma', s) \leq w_2(\sigma', s')$, that is, $|\mathcal{T}_{s_2}(\sigma')| \leq |\mathcal{T}_{s'_2}(\sigma')|$, and so $R(\sigma') \leq L(\sigma')$ (see Figure 81.4).

Likewise, since the schedule of σ is not monotone, $w_2(\sigma, s) > w_2(\sigma, s')$, and so we know that t is on machine 2 with respect to s, and t is not on machine 2 with respect to s', meaning that the length of task t contributes to the quantity $R(\sigma)$. One can deduce that $L(\sigma') < R(\sigma') + t$. Thus,

$$R(\sigma') \leq L(\sigma') < R(\sigma') + t \tag{81.6}$$

Since t is the smallest task of σ, $R(\sigma')$ and $L(\sigma')$ are either nil, or greater than or equal to t. Hence,

$$R(\sigma') \geq \frac{L(\sigma')}{2} \tag{81.7}$$

Since the schedule of σ' is monotone, while the one of σ is not monotone, we know that t is on machine 2 with respect to s, and t is on machine 1 with respect to s'. By definition of LPT, since LPT on input s assigns task t to machine 2, we know that $\frac{T_1(\sigma')+L(\sigma')+t}{s_1} \leq \frac{T_2(\sigma')+R(\sigma')+t}{s_2}$, and so

$$T_2(\sigma') \leq \frac{s_2}{s_1}(T_1(\sigma') + L(\sigma') + t) - R(\sigma') - t \tag{81.8}$$

Similarly, since LPT on input s' assigns t to machine 1, $\frac{T_1(\sigma')+R(\sigma')+t}{s'_1} \leq \frac{T_2(\sigma')+L(\sigma')+t}{s'_2}$, from which we obtain

$$T_2(\sigma') + L(\sigma') + t \geq \frac{s'_2}{s_1}(T_1(\sigma') + R(\sigma') + t) \tag{81.9}$$

since $s_1 = s'_1$. We deduce from inequalities (81.8) and (81.9) that

$$\begin{aligned}
L(\sigma') &\geq \frac{s'_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - T_2(\sigma') - t\\
L(\sigma') &\geq \frac{s'_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - \frac{s_2}{s_1}(T_1(\sigma') + L(\sigma') + t) + (R(\sigma') + t) - t\\
L(\sigma') &\geq \frac{2\, s_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - \frac{s_2}{s_1}(T_1(\sigma') + L(\sigma') + t) + R(\sigma')\\
L(\sigma') &\geq \frac{s_2}{s_1} T_1(\sigma') + \frac{s_2}{s_1}(2\, R(\sigma') + 2\, t - L(\sigma') - t) + R(\sigma')\\
L(\sigma') &\geq \frac{s_2}{s_1} T_1(\sigma') + \frac{s_2}{s_1}(2\, R(\sigma') - L(\sigma')) + t\,(\frac{s_2}{s_1} - 1) + (R(\sigma') + t)\\
L(\sigma') &\geq (R(\sigma') + t) \qquad \text{since } s_2 \geq s_1 \text{ and using inequality (81.7)}
\end{aligned}$$

This last inequality contradicts Eq. (81.6), and thus there is no instance σ for which the schedule computed by LPT is not monotone.

- The case $s_2 \geq s_1' \geq c\, s_1$ can be reduced to the previous case by observing that since the scheduling of σ with respect to s and s' is not monotone, $w_1(\sigma, s) > w_1(\sigma, s')$ and therefore $w_2(\sigma, s) < w_2(\sigma, s')$. The schedules computed by LPT with respect to s and s' are equal to the schedules computed with respect to speed vectors $\langle 1, \frac{s_2}{s_1} \rangle$ and $\langle 1, \frac{s_2}{s_1'} \rangle$, where $\frac{s_2}{s_1'} \leq c\, \frac{s_2}{s_1}$.
- Case where $s_1' > s_2$: since s_2 and s_1' are by hypothesis powers of $c \geq 2$, and since $s_1' > s_2$, then $s_1' \geq 2\, s_2$, and this case follows directly from Lemma 81.2. □

This result together with Theorem 81.8 shows that LPT is truthful for the AT model.

81.5 Other Results

In this section, we give some of the numerous results in this area classified by model. Notice the existence of other surveys [18,19] and of a recent book dealing with the price of anarchy in the context of selfish routing [20].

81.5.1 Results for the KP Model and Variants

The proof of Theorem 81.1, stating that the price of anarchy for the KP model in the case of two identical parallel machines is equal to 3/2, is due to Koutsoupias and Papadimitriou [6]. They have also pointed out the relation between the problem of computing the price of anarchy for the KP model and the BINS AND BALLS problem (see, e.g., Ref. [21]). In this way, they were able to show that the price of anarchy for the KP model with m identical machines is at least $\Omega(\frac{\log m}{\log \log m})$ and is at most $3 + \sqrt{4m \log m}$. They also obtained that for the case of the KP model with uniform machines (where the machines can have different speeds) the price of anarchy is at least $\phi = \frac{1+\sqrt{5}}{2} = 1.618$ in the case of two machines, and $O(\sqrt{\frac{s_1}{s_m} \sum_{j=1}^{m} \frac{s_j}{s_m}} \sqrt{\log m})$ in the case of m uniform machines with speeds $s_1 \geq s_2 \geq \cdots \geq s_m$. They have also conjectured that the price of anarchy for the KP model is $\Theta(\frac{\log m}{\log \log m})$ [6].

The first advance in the direction of settling the validity of this conjecture has been made by Mavronicolas and Spirakis [22], who considered a special class of Nash equilibria, the so-called *fully mixed* equilibria where for every pair of task i and machine j, p_i^j is nonzero. They proved that for this class of equilibria the conjecture of Koutsoupias and Papadimitriou is valid for the KP model with identical machines. They further considered the case of the KP model with m uniform machines and n tasks of the same length (with $m \leq n$) and they established that the price of anarchy is $\Theta(\frac{\log n}{\log \log n})$. Koutsoupias et al. [23] showed that the price of anarchy is $O(\frac{\log m}{\log \log m})$ for the KP model with identical machines. But the conjecture has been completely settled for the more general KP model with uniform machines by Czumaj and Vöcking [24], who showed that the price of anarchy is indeed $\Theta(\frac{\log m}{\log \log \log m})$.

Two important classes of Nash equilibria have been extensively studied for the KP model, namely the *pure* and the *fully mixed* Nash equilibria. The obtained results concern the evaluation of the price of anarchy, as well as the computation of the *best*, *worst*, or of *just a* Nash equilibrium within these classes.

Pure Equilibria

In the case where the agents are not allowed to randomize their strategies, the set of solutions for the KP model is the set of all *pure* Nash equilibria, that is where for every agent i and machine j, p_i^j is either 0 or 1. Several questions have been treated concerning this kind of equilibria: the first one asks if such an equilibrium always exists. It has been shown in Ref. [25] that this is true for the KP model. Another question concerns the price of anarchy when restricted to pure equilibria: Czumaj and Vöcking [24] proved two upper bounds of $\Gamma^{-1}(m) + 1 = O(\frac{\log m}{\log \log m})$ for the case of the KP model with identical machines and $O(\log \frac{s_1}{s_m})$ for the same model with uniform machines of speeds $s_1 \geq s_2 \geq \cdots \geq s_m$. They have also showed that these bounds are tight up to a constant factor. From an algorithmic point of view, the main question is how to find a Nash equilibrium: Fotakis et al. [25] showed that it is NP-hard to find the best and worst equilibria, but, they proved that it is possible to compute a Nash equilibrium for the KP model with identical machines whose price of anarchy is $\frac{4}{3} - \frac{1}{3m}$ using the LPT algorithm of Graham [9]. This result has been strengthened by the results of Refs. [26–28]. In these papers, the authors consider the *nashification problem* for the KP model, that is, the problem of designing a polynomial-time algorithm that starting from an arbitrary schedule computes a Nash equilibrium whose social cost is not greater from the one of the original schedule. Hence, in Refs. [26,27], an $O(n \log n)$ algorithm is presented for the KP model with identical machines. In Ref. [28], an $O(m^2 n)$ algorithm is presented for the KP model with uniform machines. These results show that there is a PTAS for the problem of finding a Nash equilibrium of minimum social cost for the KP model (in both the identical and uniform machine cases). This is true since it is sufficient to start from a schedule computed by the PTAS of Hochbaum and Shmoys [29] that has to be nashified using one of the above-mentioned nashification algorithms.

Fully Mixed Equilibria

As mentioned above the first to study the class of *fully mixed Nash equilibria* were Mavronicolas and Spirakis [22]. Many researchers followed this direction with the hope that the techniques elaborated for the analysis of fully mixed Nash equilibria could be appropriately extended to the general case. Fotakis et al. [25] proposed a polynomial-time algorithm, which computes in $O(n \log n)$ time a *fully mixed Nash equilibrium*. Gairing et al. [26] conjectured that the worst Nash equilibrium, that is, the Nash equilibrium with the highest social cost, is a fully mixed Nash equilibrium. This conjecture is known as the *Fully Mixed Nash Equilibrium* conjecture. The motivation to study this conjecture is that if it was true, then computing the worst Nash equilibrium would be trivial: indeed, for the KP model there is a unique fully mixed Nash equilibrium in which each task is scheduled with probability $\frac{1}{m}$ to each machine [22]. Fotakis et al. [25] studied this conjecture and gave some partial results. They proved that the conjecture is true in the special case where there are only two tasks for the KP model with identical machines. They also proved that the social cost of the worst Nash equilibrium is less than or equal to 49.02 times the cost of any generalized fully mixed Nash equilibrium for the KP model with uniform machines. They have also studied the computational complexity of computing the social cost of a Nash equilibrium, and they showed that it is #P-complete when restricted to mixed equilibria. Furthermore, they proposed a fully polynomial randomized approximation scheme (FPRAS) to compute it for the KP model with identical machines. If the fully mixed Nash equilibrium conjecture was true, this scheme would help to approximate within any accuracy the cost of the worst Nash equilibrium. Thus the efforts continued in this direction and the conjecture was shown to be true for some other special cases: the case where the number of machines is two [11] and the case where the comparison is limited to *pure* Nash equilibria [30]. Unfortunately, Fischer and Vöcking [31] showed recently that the *fully mixed Nash equilibrium* conjecture is not true and that the ratio between the social cost of a fully mixed Nash equilibrium and the worst Nash equilibrium can be almost as bad as the price of anarchy.

Various extensions of the KP model have been considered in the literature [11,33]. Among them, we can cite the *restricted* KP model where a task is allowed to be executed only to subset of the machines [11,33], the KP model with *unrelated machines* [33] or the *Web server farm* model [34]. The main results concern, as for the KP model, the evaluation of the price of anarchy or the nashification problem.

81.5.2 Results for the CKN Model

Coordination Mechanisms

The notion of coordination mechanism, the CKN model in the context of selfish scheduling, and the results of Section 81.2.2 are from Ref. [7]. In Ref. [35], the authors have recently extended the results of Ref. [7]: they noticed the close relation between coordination mechanisms and *local search algorithms*, as well as, between the price of anarchy and the *locality gap* of a neighborhood and by doing so they were able to obtain the price of anarchy for some of the variants of the CKN model using some older results from the local search literature [36]. They also studied the price of anarchy of *pure* Nash equilibria for various coordination mechanisms for the CKN model. They proved that the price of anarchy of any deterministic coordination mechanism for the CKN model with uniform machines and the CKN model where each task can be executed to a *restricted* subset of machines is at most $O(\log m)$. They also proved that the price of anarchy of a coordination mechanism based on a *universal policy* (like LPT or SPT) is $\Omega(\log m)$ for the later model. Furthermore, they showed that the price of anarchy of a coordination mechanism based on LPT is $2 - \frac{2}{m}$ for the CKN model with uniform machines. They also proved that a coordination mechanism based on a *randomized* policy has a price of anarchy of $\Theta(m)$ for the CKN model with unrelated machines. They also studied the convergence and existence of pure Nash equilibria for coordination mechanisms based on SPT and LPT policies.

Price of Stability

The results in Section 81.3.1 are from Ref. [12] and can be extended to the m machines case. The authors studied the trade-off between α and the quality of the proposed solution. More precisely, they showed that the *price of α-approximate stability* is at most $\frac{8}{7}$ for all $\alpha \geq 3$, and they gave an algorithm that achieves this bound. They also provided a relation between α and the best possible price of α-approximate stability, by showing, for example, that the price of α-approximate stability is at least $\frac{8}{7}$ for all $\alpha < 2.1$, and that it is larger than or equal to $1 + \varepsilon$ if α is smaller than a certain constant k in $\Theta(\varepsilon^{-1/2})$.

Truthfulness

The results of Section 81.4.1 are from Ref. [16]. The authors considered also the design of truthful coordination mechanisms for the CKN model with two identical machines. They first studied a *coordination mechanism* where the first machine always schedules its tasks in order of increasing lengths (its policy is SPT), and the second machine schedules its tasks with a probability $p > \frac{2}{3}$ in order of increasing lengths and with probability $(1 - p)$ in order of decreasing lengths. The expected price of anarchy of this *(randomized)* coordination mechanism, that they proved to be $\frac{4}{3} + \frac{p}{6}$, is better than the one of SPT (whose price of anarchy is $\frac{3}{2}$). They also showed that this coordination mechanism is truthful if the tasks are powers of a constant greater than or equal to $\frac{4-3p}{2-p}$, but not if the values of the task lengths are not restricted. Moreover, they showed that if $p < \frac{1}{2}$ then this coordination mechanism is not truthful even if the tasks are powers of any integer larger than 1. They also considered the other randomized coordination mechanisms that combine deterministic coordination mechanisms in which the tasks are scheduled in order of increasing or decreasing lengths (and thus which have expected price of anarchy better than the one of SPT), and gave negative results on their truthfulness.

81.5.3 Results for the AT Model

The results of Theorem 1.8 are from Ref. [8] and that of Theorem 1.9 from Ref. [17]. The first results in this direction were given by Nisan and Ronen [37], but for the model of selfish *unrelated* machines. For this model they proved that the former known approximation algorithms are not truthful. For the AT model (with uniform machines) it was Archer and Tardos who introduced a truthful *randomized* mechanism, which gives 3-approximate solutions. The first deterministic result is due to Auletta et al. [38] who proposed a deterministic polynomial-time $(2 + \varepsilon)$-approximation algorithm and suitable payments functions that yield truthful mechanisms for the following restrictions of the problem: (i) the speeds of the machines are integer and the largest is bounded from above by a constant, and (ii) the speeds are divisible, that is, s_{i+1} is a multiple of s_i. The proposed mechanisms compute the payments in polynomial time and satisfy *voluntary participation*, that is, a truthfully behaving agent never incurs in a loss. They were also able to deduce a deterministic truthful $(4 + \varepsilon)$-approximate mechanism for the case of *arbitrary speeds* and for any, but *fixed*, number of machines. More recently, Azar and Sorani [39] improved these results: They provided a deterministic 12-approximation truthful mechanism for the AT model with an *arbitrary* number of processors. They also proposed a deterministic truthful PTAS for the AT model with a *fixed* number of machines. Auletta et al. [40] studied the difficulty of translating approximation/competitive algorithms into equivalent approximation/competitive truthful mechanisms. They proposed "translation" technique and studied its limits.

References

[1] Papadimitriou, C., Algorithms, games and the Internet, *Proc. STOC*, 2001, p. 749.

[2] Nash, J. F., Non cooperative games, *Ann. Math.*, 54, 286, 1951.

[3] Osborne, J. O. and Rubinstein, A., A *Course in Game Theory*, The MIT Press, Cambridge, MA, 1994.

[4] Lipton, R. J., Markakis, E., and Mehta, A., Playing large games using simple strategies, *Proc. 4th ACM Conf. on Electronic Commerce,* 2003, p. 36.

[5] Papadimitriou, C. and Roughgarden, T., Computing equilibria in multi-player games, *Proc. SODA*, 2005, p. 82.

[6] Koutsoupias, E. and Papadimitriou, C., Worst-case equilibria, *Proc. STACS*, Lecture Notes in Computer Science, Vol. 1563, 1999, Springer, Berlin, p. 404.

[7] Christodoulou, G., Koutsoupias, E., and Nanavati, A., Coordination mechanisms, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 3142, Springer, Berlin, 2004, p. 345.

[8] Archer, A. and Tardos, E., Truthful mechanisms for one-parameter agents, *Proc. FOCS*, 2001, p. 482.

[9] Graham, R., Bounds on multiprocessor timing anomalies, *SIAM J. Appl. Math.*, 17(2), 416, 1969.

[10] Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, É., Wexler, T., and Roughgarden, T., The price of stability for network design with fair cost allocation, *Proc. FOCS*, 2004, p. 295.

[11] Gairing, M., Lücking, T., Mavronicolas, M., and Monien, B., Computing Nash equilibria for scheduling on restricted parallel links, *Proc. STOC,* 2004, p. 613.

[12] Angel, E., Bampis, E., and Pascual, F., The price of approximate stability for a scheduling game problem, Euro-Par, LNCS, 2006.

[13] Clarke, E., Multipart pricing of public goods, *Public Choices*, 11, 17, 1971.

[14] Groves, T., Incentive in teams, *Econometrica*, 41(4), 617, 1973.

[15] Vickrey, W., Counterspeculation, auctions and competitive sealed tenders, *J. Finance*, 16, 8, 1961.

[16] Angel, E., Bampis, E., and Pascual, F., Truthful algorithms for scheduling selfish tasks on parallel machines, *Proc. WINE*, Lecture Notes in Computer Science, Vol. 3828, Springer, Berlin, 2005, p. 698.

[17] Ambrosio, P. and Auletta, V., Deterministic monotone algorithms for scheduling on related machines, *Proc. WAOA*, Lecture Notes in Computer Science, Vol. 3351, Springer, Berlin, 2004, p. 267.

[18] Czumaj, A., Selfish routing on the Internet, in *Handbook of Scheduling: Algorithms, Models and Performance Analysis,* Leung, J., Ed., CRC Press, Boca Raton, FL, 2004, chap. 42.

[19] Feldmann, R., Gairing, M., Lücking, T., Monien, B., and Rode, M., Selfish routing in non-cooperative networks: a survey, *Proc. MFCS*, Lecture Notes in Computer Science, Vol. 2747, Springer, Berlin, 2003, p. 21.

[20] Roughgarden, T., Selfish Routing and the Price of Anarchy, The MIT Press, Cambridge, MA, 2005.

[21] Gonnet, G., Expected length of the longest probe sequence in hash code searching, *JACM*, 28(2), 289, 1981.

[22] Mavronicolas, M. and Spirakis, P., The price of selfish routing, *Proc. STOC*, 2001, p. 510.

[23] Koutsoupias, E., Mavronicolas, M., and Spirakis, P., Approximate equilibria and ball fusion, *Theor. Comput. Syst.*, 36(6), 683, 2003.

[24] Czumaj, A. and Vöcking, B., Tight bounds for worst-case equilibria, *Proc. SODA*, 2002, p. 413.

[25] Fotakis, D., Kontogiannis, S., Koutsoupias, E., Mavronicolas, M., and Spirakis, P., The structure and complexity of Nash equilibria for a selfish routing game, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 2380, Springer, Berlin, 2002, p. 123.

[26] Gairing, M., Lücking, T., Mavronicolas, M., Monien, B., and Spirakis, P., Extreme Nash equilibria, *Proc. ICTCS*, Lecture Notes in Computer Science, Vol. 2841, Springer, Berlin, 2003, p. 1.

[27] Even-Dar, E., Kesselman, A., and Mansour, Y., Convergence time to Nash equilibria, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 2719, Springer, Berlin, 2003, p. 502.

[28] Feldmann, R., Gairing, M., Lücking, T., Monien, B., and Rode, M., Nashification and the coordination ratio for a selfish routing game, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 2719, Springer, Berlin, 2003, p. 514.

[29] Hochbaum, D. and Shmoys, D., A polynomial approximation scheme for scheduling in uniform processors: using the dual approximation scheme, *SIAM J. Comput.*, 17(3), 539, 1998.

[30] Gairing, M., Lücking, T., Mavronicolas, M., Monien, B., and Spirakis, P., Structure and complexity of extreme Nash equilibria, *Theor. Comp. Sci.*, 343(1–2), 133, 2005.

[31] Fischer, S. and Vöcking, B., On the structure and complexity of worst-case equilibria, *Proc. WINE,* Lecture Notes in Computer Science, Vol. 3828, Springer, Berlin, 2005, p. 151.

[32] Ferrante, A. and Parente, M., Existence of Nash equilibria in selfish routing problems, *Proc. SIROCCO,* Lecture Notes in Computer Science, Vol. 3104, Springer, Berlin, 2004, p. 149.

[33] Awerbuch, B., Azar, Y., Richter, Y., and Tsur, D., Tradeoffs in worst-case equilibria, *Proc. WAOA*, Lecture Notes in Computer Science, Vol. 2909, Springer, Berlin, 2003, p. 64.

[34] Czumaj, A., Krysta, P., and Vöcking, B., Selfish traffic allocation for server farms, *Proc. STOC*, 2002, p. 287.

[35] Immorlica, N., Li, L., Mirrokni, V. S., and Schulz, A., Coordination mechanisms for selfish scheduling, *Proc. WINE*, Lecture Notes in Computer Science, Vol. 3828, Springer, Berlin, 2005, p. 55.

[36] Vredeveld, T., Combinatorial Approximation Algorithms. Guaranteed versus Experimental Performance, Ph.D. thesis, Technische Universiteit Eindhoven, The Netherlands, 2002.

[37] Nisan, N. and Ronen, A., Algorithmic mechanism design, *Proc. STOC*, 1999, p. 129.

[38] Auletta, V., De Prisco, R., Penna, P., and Persiano, P., Deterministic truthful approximation mechanisms for scheduling related machines, *Proc. STACS*, Lecture Notes in Computer Science, Vol. 2996, Springer, Berlin, 2004, p. 608.

[39] Azar, Y. and Sorani, M., Truthful approximation mechanisms for scheduling selfish related machines, *Proc. STACS*, Lecture Notes in Computer Science, Vol. 3404, Springer, Berlin, 2005, p. 69.

[40] Auletta, V., De Prisco, R., Penna, P., and Persiano, P., On designing truthful mechanisms for online scheduling, *Proc. SIROCCO*, Lecture Notes in Computer Science, Vol. 3499, Springer, Berlin, 2005, p. 3.

82

Approximate Economic Equilibrium Algorithms

Xiaotie Deng
City University of Hong Kong

Li-Sha Huang
Tsinghua University

82.1 Introduction

The general equilibrium model of the economy consists of a set of agents, each with an initial endowment of commodities, interacting through a market, trying to maximize their own utility function. The market prices of commodities are determined by a clearance condition. That is, all commodities are bought, collectively, by all the utility maximizing agents, subject to their budget constraints (determined by the values of their initial endowments of commodities at the market price).

This conceptual framework is the outcome of a sequence of efforts trying to fully understand the laws that govern human commercial activities, starting with the "invisible hand" of Adam Smith [1], the Walras law [2], and finally, the mathematical conclusion of Arrow and Debreu [3] that there exists a set of prices that bring supply and demand into equilibrium, under quite general conditions on the agent utility functions and their optimization behavior.

An inherent challenge grown out of the mathematical beauty of the general equilibrium, especially to the believers of the model often referred to as Neoclassical Economics, is how this clearance price vector arrives at the market place. Walras proposed a tatonnement process that moves the market into an equilibrium state through a sequence of virtual auctions of the commodities at a sequences of their prices that gradually correct imbalance of the supply and the demand. In a different approach, Scarf developed a fixed point algorithm to directly compute the equilibrium price [4]. In addition, mathematical programming models and numerical methods have been used in practical methods commonly referred to as the paradigm of Computable General Equilibrium (CGE) [5]. Those approaches can be traced back to the Leontief's input–output model and have been applied to policy and trade analysis of developing countries, most noticeably, by the World Bank [6,7].

The Scarf's fixed-point algorithm approach was the first serious effort on deriving the equilibrium price for general utility functions. Scarf observed that the algorithm converges in $O(n^4)$ time. The worst-case complexity, however, has shown to be exponential [8,9]. The fixed-point algorithm does not reach an exact solution of the general equilibrium problem but an approximate solution in the fixed-point formulation. It may not even directly translate to an approximate general equilibrium solution in a well defined way.

For practical applications, solutions are always approximate as various parameters are not as accurate as in an ideal world. Data are estimated and utility functions are derived. Therefore, the fact that the exact solutions cannot be efficiently computed has not been a major concern for any practical reason. Conceptually, however, approximate general equilibrium has never hold a place in the mainstream Economics Theory, except in special cases, such as the case for the rational expectation model, where Allen [10] defined approximation equilibrium as a utility maximization model with the aggregated excess demand bounded by ϵ. This concept has thus rescued the general equilibrium approach for the rational expectation model, which may result in nonexistent of the exact equilibrium.

In a proposal for computational complexity study of the general equilibrium problem, Deng et al. [11] introduced a concept of approximate general equilibrium in a more relaxed way than that of Allen. They introduce the computational counter part of bounded rationality, that of Herbert Simon [12], to allow agents in the market to be satisfied with allocations of commodities with a utility within $1 - \epsilon$ of its optimal utility under the price, together with the global excess demand relaxation as that of Allen.

The approximate equilibrium concept allows them to develop polynomial time approximate solution for the case of a finite number of indivisible goods. Even for divisible goods where equilibrium always exist, the concept has allowed for polynomial time approximate solutions under various models of the economy [13,14]. Moreover, the concept has formed a basis for the study of an interesting dynamic model of the online market [15].

The computer and Internet age has created a reality of the economy much different from that led to the establishment of the theories of Smith, Walras, and even Arrow and Debreu. Economic activities become observable in a wide scale that matches up the considerations under the general equilibrium. The Electronic Commerce and the Internet have made the ideal testing ground for the then speculative theories of the prophets. Economics can become as empirical as any scientific subjects. The recent study in this direction is only the beginning of a great revolution to come in Economic Theories.

In this chapter, we will discuss recent development for the algorithmic complexity issue of the general equilibrium problem. We should focus on pure exchange economy for simplicity of presentation. The general model with production agents can often be handled similarly. In Section 82.2, we first introduce the model of general equilibrium, and define the notation of approximate equilibrium, together with some discussion on the recent development in this direction. In Sections 82.3 and 82.4, we discuss two algorithmic approaches for computational of the general equilibrium problem, including convex programming and ellipsoid algorithms. In Section 82.5, we discuss approaches that derive provably approximate algorithms for the general equilibrium problem, especially with indivisible commodities in the market. In Section 82.6, we discuss the hardness results for approximating the market equilibria. We conclude in Section 82.7 with remarks on open problems and future directions.

82.2 Models and Definitions

In the section, we will introduce the model of a pure exchange economy and the definition of equilibrium of the economy.

In a pure exchange economy, there are m traders, labelled by $i = 1, 2, \dots, m$, and n types of commodities, labelled by $j = 1, 2, \dots, n$. The commodities could be divisible or indivisible. Each trader i comes to the market with initial endowment of commodities, denoted by a vector $w_i \in \mathbb{R}^n_+$, whose jth entry is the amount of commodity j held by trader i.

We associate each trader i a *consumption set* X_i to represent the set of possible commodity bundles. For example, when there are n_1 divisible commodities and $(n - n_1)$ indivisible commodities, we may set $X_i = \mathbb{R}^{n_1}_+ \times \mathbb{Z}^{n-n_1}_+$. Each trader has a utility function $X_i \mapsto \mathbb{R}_+$ to present his utility for a bundle of commodities. Usually, we require that the utility function u be concave and nondecreasing.

Example 82.1

Constant elasticity of substitution (CES) function has the form $u(x) = \left(\sum_{j=1}^{n} \alpha_j x_j^{\rho}\right)^{\frac{1}{\rho}}$, where $\alpha'_j s$ are constant parameters, and $\rho \in (-\infty, 1] \backslash \{0\}$. CES function is intensively used in economic modelling of markets. It also covers some important classes of utility functions, at least in the limit sense. For example, CES function turns to be linear utility function when $\rho = 1$, Cobb–Douglas utility function when $\rho \to 0$, Leontief utility function when $\rho \to -\infty$.

In the market, each trader acts as both a buyer and a seller to maximize his utility. At a certain price $p \in \mathbb{R}_+^n$, trader i is solving the following optimization problem, under his budget constraint:

$$\max \quad u_i(x_i) \quad \text{s.t.} \quad x_i \in X_i \text{ and } \langle p, x_i \rangle \le \langle p, w_i \rangle$$

Definition 82.1

An equilibrium in a pure exchange economy is a price vector $\bar{p} \in \mathbb{R}_+^n$ and bundles of commodities $\left\{\bar{x}_i \in \mathbb{R}_+^n, i = 1, \ldots, m\right\}$, such that

$$\bar{x_i} \in argmax\{u_i(x_i)|x_i \in X_i \text{ and} \langle x_i, \bar{p}\rangle \le \langle w_i, \bar{p}\rangle\}, \forall 1 \le i \le m$$
$$\sum_{i=1}^{m} \bar{x}_{ij} \le \sum_{i=1}^{m} w_{ij}, \forall 1 \le j \le n$$

A special case of exchange market is the Fisher's model which can be considered as a special case of the general model, where the initial endowments of traders are proportional, i.e., $w_i = e_i w$ to a fixed vector $w \in \mathbb{R}_+^n$. In fact, Fisher considered the traders who come to the market with initial endowments of money, where the goods are available for sale. The traders buy commodities from the market to maximize their utilities under their budget constraints. Assume trader i's money is $e_i \in \mathbb{R}_+$, then he is solving the following optimization problem:

$$\max u_i(x_i) \quad s.t. \quad x_i \in X_i \text{ and } \langle p, x_i \rangle \le e_i$$

The equilibrium price for the Fisher model is one under which all traders spent all their money and all goods are sold to the traders.

Definition 82.2

An equilibrium in the Fisher's model is a price vector $\bar{p} \in \mathbb{R}_+^n$ and bundles of commodities $\left\{\bar{x}_i \in \mathbb{R}_+^n, i = 1, \ldots, n\right\}$, such that

$$\bar{x_i} \in argmax\{u_i(x_i)|x_i \in X_i \text{ and} \langle x_i, \bar{p}\rangle \le e_i\}, \forall 1 \le i \le m$$
$$\sum_{i=1}^{m} \bar{x}_{ij} \le \sum_{i=1}^{m} e_i w, \forall 1 \le j \le n$$
$$\left\langle \bar{p}, \sum_{i=1}^{m} \bar{x}_i \right\rangle = \left\langle \bar{p}, \sum_{i=1}^{m} e_i w \right\rangle$$

Even though an equilibrium exists under some moderate assumptions, its computation is not in general easy. The classical general approach of the fixed-point method is known to be exponential in time. In addition, existence is not always guaranteed for indivisible goods, even for divisible goods under some utility functions. Therefore, a concept of approximate equilibrium was introduced in Ref. [11]:

Definition 82.3 [11]

An ϵ-approximate equilibrium in an exchange market is a price vector $\bar{p} \in \mathbb{R}_+^n$ and bundles of goods $\left\{\bar{x}_i \in \mathbb{R}_+^n, i = 1, \ldots, m\right\}$, such that

$$u_i(\bar{x}_i) \ge \frac{1}{1+\epsilon} max\{u_i(x_i)|x_i \in X_i, \langle x_i, \bar{p}\rangle \le \langle w_i, \bar{p}\rangle\}, \forall i \qquad (82.1)$$

$$\langle \bar{x}_i, \bar{p} \rangle \le \langle w_i, \bar{p} \rangle, \forall i \tag{82.2}$$

$$\sum_{i=1}^{m} \bar{x}_{ij} \ge (1-\epsilon) \sum_{i=1}^{m} w_{ij}, \forall j \tag{82.3}$$

$$\sum_{i=1}^{m} \bar{x}_{ij} \le \sum_{i=1}^{m} w_{ij}, \forall j \tag{82.4}$$

82.3 Convex Programming Methods

Convex programming is a powerful tool to solve the equilibrium problem. By exploiting the Karush-Kuhn-Tucker (KKT) conditions of the traders' utility-maximizing problem, the equilibrium conditions may be formulated as a convex feasibility problem or a convex optimization problem. Such convex formulations are expressive to characterize many utility functions, and are also implementable in practice. Up till now, at least two classes of problems can be solved with this kind of methods. We will introduce them in the following two subsections.

82.3.1 A Convex Formulation for a Class of General Equilibrium Model

In 1983, Nenakhov and Primak [16] found that in an exchange market with linear utility functions, the equilibrium conditions are equivalent to a set of convex feasibility conditions. Recently, Jain [17] rediscovered this convex feasibility formulation independently. Since then, the method has been applied to more general cases including productivity components as by Jain et al. [18]

The following theorem is a simplified version of Nenakhov and Primak's [16] convex formulation (with the production activities removed). It provides a sufficient and necessary condition for the equilibrium in a pure exchange market:

Theorem 82.1

Let u_{ij} denote $\frac{\partial u_i}{\partial j}$. An allocation-price pair (x, p) is an equilibrium if and only if

$$\frac{u_{ij}(x_i)}{p_j} \le \frac{\langle \nabla u_i, x_i \rangle}{\langle w_i, p \rangle}, \forall i, j. \tag{82.5}$$

$$\sum_{i=1}^{m} x_{ij} \le \sum_{i=1}^{m} w_{ij}, x_{ij} \ge 0, \forall i, j \tag{82.6}$$

Proof

Given a price vector p, trader i is solving the following optimization problem:

$$\max \quad u_i(x_i) \tag{82.7}$$

$$\text{s.t.} \quad f_0(x_i) = \langle x_i, p \rangle - \langle p, w_i \rangle \le 0 \tag{82.8}$$

$$f_j(x_i) = -x_{ij} \le 0, \forall 1 \le j \le n \tag{82.9}$$

And its dual problem:

$$\min \quad u_i(x_i) - \sum_{k=0}^{n} y_k f_k(x_i) \tag{82.10}$$

$$\text{s.t.} \quad \sum_{k=0}^{n} y_k \nabla f_k(x_i) = \nabla u_i(x_i) \tag{82.11}$$

$$y_k \ge 0, \forall k \tag{82.12}$$

It is known that x_i is the optimal solution of the primal problem if and only if there exists a $y \in \mathbb{R}^{m+1}$ such that

$$\begin{aligned} y_0(\langle x_i, p\rangle - \langle p, w_i\rangle) - \sum_{k=1}^{n} y_k x_{ik} &= 0 \\ \langle x_i, p\rangle &\leq \langle p, w_i\rangle \\ y_0 p_j - y_j &= u_{ij}(x_i), \forall j \\ x_{ij} &\geq 0, \forall j \\ y_k &\geq 0, \forall k \end{aligned}$$

Take $y_j = y_0 p_j - u_{ij}(x_i)$, the conditions are equivalent to that there exists (x_i, y_0), such that

$$y_0 \langle p, w_i\rangle = y_0 \langle x_i, p\rangle \tag{82.13}$$
$$\langle x_i, p\rangle \leq \langle p, w_i\rangle \tag{82.14}$$
$$y_0 p_j \geq u_{ij}(x_i), \quad \forall 1 \leq j \leq n \tag{82.15}$$
$$y_0 p_j x_{ij} = u_{ij}(x_i) x_{ij}, \quad \forall 1 \leq j \leq n \tag{82.16}$$
$$x_{ij} \geq 0, \quad \forall 1 \leq j \leq n \tag{82.17}$$
$$y_0 \geq 0 \tag{82.18}$$

Now we prove the necessary side. If (x, p) is an equilibrium, then for any x_i, we can find a y_0 satisfies Eqs. (82.13)–(82.18). By Eq. (82.13) and Eq. (82.16), we have

$$\begin{aligned} \langle \nabla u_i, x_i\rangle = \sum_{j=1}^{n} u_{ij}(x_i) x_{ij} &= \sum_{j=1}^{n} y_0 p_j x_{ij} \\ &= y_0 \langle x_i, p\rangle = y_0 \langle p, w_i\rangle \end{aligned}$$

Since $y_0 \geq \frac{u_{ij}}{p_j} (\forall j)$, we have proved that $\langle \nabla u_i, x_i\rangle / \langle p, w_i\rangle \geq u_{ij}(x_i)/p_j$ for all i, j.

For the sufficient side, if there exists (x, p) such that

$$\begin{aligned} \frac{u_{ij}(x_i)}{p_j} &\leq \frac{\langle \nabla u_i, x_i\rangle}{\langle w_i, p\rangle}, \forall i, j. \\ \sum_{i=1}^{m} x_{ij} &\leq \sum_{i=1}^{m} w_{ij}, \quad \forall j; \\ x_{ij} &\geq 0, \quad \forall i, j \end{aligned}$$

We have

$$\begin{aligned} &\langle \nabla u_i, x_i\rangle\, p_j \geq u_{ij} \langle p, w_i\rangle \\ &\Rightarrow \sum_{j=1}^{n} \langle \nabla u_i, x_i\rangle\, p_j x_{ij} \geq \sum_{j=1}^{n} u_{ij} x_{ij} \langle p, w_i\rangle \\ &\Rightarrow \langle x_i, p\rangle \geq \langle p, w_i\rangle \end{aligned}$$

Let $\langle x_i, p\rangle = \langle p, w_i\rangle + \epsilon_i$. Sum over i, we have

$$\left\langle \sum_{i=1}^{n} x_{ij}, p \right\rangle = \left\langle \sum_{i=1}^{n} w_{ij}, p \right\rangle + \sum_{i=1}^{n} \epsilon_i$$

Then, the condition $\sum_{i=1}^{m} x_{ij} \leq \sum_{i=1}^{m} w_{ij}$ implies that $\epsilon_i = 0$ for all i. Now, x_i have satisfied conditions (82.13), (82.14), and (82.17).

For any i, take $y_0 = \max_j \frac{u_{ij}}{p_j}$, then we need only to verify Eq. (82.16). We have for all j:

$$\begin{aligned} & y_0 p_j x_{ij} \geq u_{ij}(x_i) x_{ij} \\ & \Rightarrow \sum_{j=1}^{n} y_0 p_j x_{ij} \geq \sum_{j=1}^{n} u_{ij}(x_i) x_{ij} \\ & \Rightarrow y_0 \langle x_i, p \rangle \geq \langle \nabla u_i, x_i \rangle \end{aligned}$$

Finally, we get $y_0 \langle p, w_i \rangle \geq \langle \nabla u_i, x_i \rangle \geq y_0 \langle p, w_i \rangle$. So all inequalities become equalities. Hence, $y_0 p_j x_{ij} = u_{ij}(x_i) x_{ij}$ for all j. This complete the proof. □

Corollary 82.1

By Theorem 82.1, the problem of computing a general equilibrium can be reformulated to:

$$\begin{aligned} & \min \quad \theta \\ & \text{s.t.} \quad \sum_{i=1}^{m} x_{ij} = \sum_{i=1}^{m} w_i + \theta, \forall j \\ & \log\left(\frac{\langle \nabla u_i, x_i \rangle}{u_{ij}}\right) \geq \log(\langle p, w_i \rangle) - \log(p_j), \forall i, j \\ & x_{ij} \geq 0, \ p_j > 0, \forall i, j \end{aligned} \tag{82.19}$$

The following lemma was developed in Ref. [19]:

Lemma 82.1

For all feasible solution of Problem (82.19), we must have $\theta \geq 0$.

Proof

The inequalities in (82.19) imply that

$$\begin{aligned} & x_{ij} p_j \langle \nabla u_i, x_i \rangle \geq \langle p, w_i \rangle u_{ij} x_{ij}, \forall i, j \\ (\text{summing over } j) \Rightarrow & \langle x_i, p \rangle \langle \nabla u_i, x_i \rangle \geq \langle p, w_i \rangle \langle \nabla u_i, x_i \rangle \\ \Rightarrow & \langle x_i, p \rangle \geq \langle p, w_i \rangle \\ (\text{summing over } i) \Rightarrow & \left\langle \sum_{i=1}^{m} x_i, p \right\rangle \geq \left\langle \sum_{i=1}^{m} w_i, p \right\rangle \end{aligned}$$

This implies $\theta \geq 0$. □

The existence of equilibrium and Lemma 82.1 show that the minimal value of Problem (82.19) must be zero. If $\log(\frac{\langle \nabla u_i, x_i \rangle}{u_{ij}})$ is concave for any i, j, the system (Eq. (82.19)) is a convex optimization problem. Therefore, we can approximate the equilibrium in polynomial time with existing techniques for convex programming, such as interior point algorithms. The time complexity strongly depends on the utility functions. Ye [19] shows how to estimate the time complexity for linear utility functions.

82.3.2 Homogeneous Utility Functions

It was first discovered by Eisenberg and Gale [20] that the solution of a convex optimization program yields the market equilibrium in the Fisher's market model with linear utility functions. Later, Eisenberg [21] extends the approach to concave homogeneous utility functions. In this subsection, we redescribe their result in a general form. We will show that how to aggregate the traders' individual utilities to obtain a convex optimization program and how the equilibrium price be yielded by the solution of the program.

In this subsection, we deal with the Fisher's market model. In the market, the money held by trader i is $e_i \in \mathbb{R}_+$ and the amount of each commodity is normalized to 1.

A continuous differentiable function $u : \mathbb{R}_+^n \mapsto \mathbb{R}_+$ is said to be *homogenous* of degree d if $\langle \nabla u, x \rangle = du(x)$. CES utility functions are homogenous functions.

A continuous differentiable function $u : \mathbb{R}_+^n \mapsto \mathbb{R}_+$ is said to be *weak homogenous* if there exists an increasing function $\phi : \mathbb{R}_+ \mapsto \mathbb{R}_+$ such that

$$\langle \nabla u, x \rangle = \phi(u) \tag{82.20}$$

Obviously, a homogenous function of degree d is a weak homogenous function with $\phi(u) = du$.

Assume trader i's utility function u_i is weak homogenous, then there exists a function ϕ_i satisfying that

$$\langle \nabla u_i, x_i \rangle = \phi_i(u_i(x_i))$$

Define a function $\Phi_i : \mathbb{R}_+ \mapsto \mathbb{R}_+$ by the integration of ϕ_i

$$\Phi_i(u) = \int\limits_{t=0}^{u} \frac{e_i}{\phi_i(t)} dt$$

Naturally, $d\Phi_i(u_i(x_i)) = \frac{e_i}{\phi(u_i(x_i))} = \frac{e_i}{\langle \nabla u_i, x_i \rangle}$. Since ϕ_i is an increasing function, Φ_i is a well-defined concave function over $\mathbb{R}_+$. For example, when u_i is a homogenous function of degree d, $\Phi(u_i) = e_i \log(u_i)$.

Theorem 82.2

If the utility functions of all the traders satisfy the weak homogenous condition, then the market equilibrium in the Fisher's setting can be computed by solving the following convex programming problem:

$$\begin{aligned} \max \quad & \Phi = \sum_{i=1}^{m} \Phi_i(u_i(x_i)) \\ \text{s.t.} \quad & \sum_{i=1}^{m} x_{ij} \leq 1, \forall j \\ & x_{ij} \geq 0, \forall i, j \end{aligned} \tag{82.21}$$

Proof

Let x be the solution of Eq. (82.21) and p be the dual solution. Denote $\frac{\partial u_i}{\partial x_{ij}}$ by u_{ij}, then

$$\begin{aligned} & \tfrac{\partial \Phi}{\partial x_{ij}} = d\Phi_i(u_i)u_{ij} \leq p_j, \forall i, j \\ & d\Phi_i(u_i)u_{ij}x_{ij} = p_j x_{ij} \forall i, j \\ & \sum_{i=1}^{m} x_{ij} \leq 1, \forall j \\ & p_j \left(1 - \sum_{i=1}^{m} x_{ij}\right) = 0, \forall j \\ & x_{ij} \geq 0, \forall i, j \end{aligned}$$

These equations guarantee that x_i is the solution of trader i's utility-maximizing problem

$$\begin{aligned} \max \quad & u_i(x_i) \\ \text{s.t.} \quad & \langle x_i, p \rangle \leq e_i \end{aligned}$$

To see p is a market clearing price, we have for any trader i:

$$\begin{aligned} \sum_{j=1}^{n} p_j x_{ij} &= \sum_{j=1}^{n} d\Phi_i(u_i)u_{ij}x_{ij} = \sum_{j=1}^{n} \frac{e_i}{\phi_i(u_i)} u_{ij}x_{ij} \\ &= \frac{e_i}{\langle \nabla u_i, x_i \rangle} \sum_{j=1}^{n} u_{ij}x_{ij} = e_i \end{aligned}$$

□

82.4 Ellipsoid Algorithm

In some models, it is more convenient to use demand functions to characterize the traders' preference on goods. Trader i's demand function is a map $d_i : \mathbb{R}^n_+ \mapsto \mathbb{R}^n_+$, where $d_i(p) \in \mathbb{R}^n_+$ presents trader i's demand of goods under the price p. The excess demand function of trader i is $z_i(p) = d_i(p) - w_i$. The aggregated demand function is defined by $D(p) = \sum_{i=1}^{mj} d_i(p)$ and the aggregated excess demand function is $Z(p) = D(p) - \sum_{i=1}^{m} w_i$.

We can define the market equilibrium price via excess demand functions.

Definition 82.4

A price p is an equilibrium price of the market if and only if $Z(w, p) \leq 0$.

If the trader's utility function $u_i : \mathbb{R}^n_+ \mapsto \mathbb{R}_+$ is strictly concave, we can define its demand function $d_i : \mathbb{R}^n_+ \mapsto \mathbb{R}^n_+$ as follows:

$$d_i(p) = argmax\left\{u_i(x_i) \mid \langle x_i, p\rangle \leq \langle w_i, p\rangle, x_i \in \mathbb{R}^n_+\right\}$$

If the traders' demand functions are induced by their utility functions, Definition 82.1 and Definition 82.4 are equivalent.

We can also define the approximate equilibrium via the excess demand function.

Definition 82.5

A price vector p and bundles of goods $\{x_i \mid i = 1, \ldots, m\}$ is a strong ϵ-approximate equilibrium if $x_i = d_i(p)$ for every i and $\sum_{i=1}^{m} x_{ij} \leq (1+\epsilon)\sum_{i=1}^{m} w_{ij}$.

The demand functions may have the following properties:

1. A demand function d_i is said to satisfy the Walras Law, if $\langle p, d_i(p)\rangle = \langle p, w_i\rangle$ for any price p, or equivalently, $\langle p, z_i(p)\rangle = 0$.
2. A demand function d_i is said to satisfy positive homogeneity, if $d_i(p) = d_i(\lambda p)$ for any price p and any $\lambda > 0$.
3. A demand function d_i is said to satisfy weak gross substitutability (WGS) if any two prices p and p' such that $0 < p_j \leq p'_j$ for each j, and $p_j < p'_j$ for some j, we have that $p_k = p'_k$ implies $d_{i,k}(p) \leq d_{i,k}(p')$.

Arrow et al. [22] proved the following lemma.

Lemma 82.2 (Separation Lemma)

If the equilibrium vector $\bar{p} > 0$ and gross substitutability prevails and the Walras Law together with positive homogeneity hold, then for any nonequilibrium vector $p > 0$, we have

$$\langle \bar{p}, Z(p)\rangle \equiv \sum_{i=1}^{n} \bar{p}_i Z_i(p) > 0 \tag{82.22}$$

By the Walras law, $\langle \bar{p} - p, Z(p)\rangle = \langle \bar{p}, Z(p)\rangle > 0$. This inequality shows that there always exists a hyperplane that separates an nonequilibrium price vector from the set of equilibrium prices. The separation lemma provides an intuition that we may develop an ellipsoid algorithm to find an equilibrium price, with an oracle to compute the aggregate excess demand function. However, the inequality (Eq. (82.22)) is not enough to guarantee the polynomial-time convergence of the ellipsoid algorithm, since it does not provide a lower bound on the distance between the equilibrium price and the separate hyperplane.

Codenotti et al. [23] first exploit the separation lemma to compute the equilibrium price. They present a polynomial-time ellipsoid algorithm to compute a weak ϵ-approximate equilibrium, when the aggregated excess demand function satisfies the Walras law, positive homogeneity, and WGS. (A weak approximate equilibrium is similar to Definition 82.5, except that the demand functions are also approximated).

They also show that if $\|\frac{\partial Z}{\partial p}\|_\infty$ is polynomial-bounded, the ellipsoid algorithm can compute a strong ϵ-approximate equilibrium in polynomial time.

The content of this section is based on Codenotti et al.'s work [23]. To avoid too much details, we only introduce their result on computing the strong approximate equilibrium, and assume the market satisfies the following two assumptions:

Assumption 1

For any equilibrium price p, $max\{p_j\}/min\{p_j\} \leq 2^L$, where D is bounded by a polynomial in the input size.

Assumption 1 is reasonable because if the market does not satisfy this assumption, we can add a virtual trader to the market, whose utility function is the Cobb–Douglas function $u_{m+1}(x_{m+1}) = \prod_{j=1}^n x_{m+1,\,j}^{1/n}$, and whose endowment is $(\eta, \eta, \ldots, \eta)$ for a small number $\eta > 0$. This trader will prevent the smallest price from being too small relative to the largest price.

By the positive homogeneity, if p is an equilibrium price, so is λp for any $\lambda > 0$. Therefore, we can assume that the domain of the demands functions is the region $\Delta = \{p \in \mathbb{R}_+^n \mid 2^{-L} \leq p_j \leq 1, \forall j\}$.

Assumption 2

$|\frac{\partial Z_i}{\partial p_j}(p)| \leq 2^D$ for any i, j and any $p \in \Delta$, where D is bounded by a polynomial in the input size.

Lemma 82.3 **(Enhanced Separation Lemma)** [23]

Assume that the market M satisfies the Walras law, positive homogeneity, WGS, Assumption 1, and Assumption 2. Let $p \in \Delta$ be a price vector that is not a strong ϵ-approximate equilibrium, for some $\epsilon > 0$. Then for any equilibrium $\bar{p} \in \Delta$, we have $\langle \bar{p}, Z(p)\rangle \geq \delta$, where $\delta \geq 2^{-E_1}$, and E_1 is bounded by a polynomial in the input size of M and $\log(1/\epsilon)$. *Moreover, $\|Z(p)\|_2 \leq 2^{E_2}$, where E_2 is bounded by a polynomial in the input size.*

Before proving Lemma 82.3, we first show the algorithmic implication of the enhanced separation lemma by presenting an inscribed ellipsoid algorithm for computing a strong ϵ-approximate equilibrium. The inscribed ellipsoid algorithm was proposed by Huang et al. [24] for computing the fixed-point of a nonexpansive map, and is also suitable for our case.

For a convex set $A \subset \mathbb{R}^n$, we define $\mu(A)$ by

$$\mu(A) = \max\{\text{the volume of } E \mid E \text{ is an ellipsoid and } E \subseteq A\}.$$

We call an ellipsoid $E \subseteq A$ is γ-maximal if $Vol(E) \geq \gamma\mu(A)$. Now we can formulate the ellipsoid algorithm as follows:

```
Ellipsoid Algorithm
Input:  A market $M$ and an oracle of its aggregate excess demand function, $\epsilon > 0$.
Output:  A strong $\epsilon$-approximate equilibrium price of $M$.
Step 0:  Set $\gamma = 0.99$. Set $i = 0$ and $A_0 = \Delta$.
Step 1:  Construct a $\gamma$-maximal ellipsoid $E_i$ in $A_i$ and let $p^{(i)}$ be its center.
Step 2:  If $p^{(i)}$ is a strong $\epsilon$-approximate equilibrium, terminate with the output $p^{(i)}$.
Step 3:  Let $A_{i+1} = \{y \in A_i \mid \langle Z(p^{(i)}), y - p^{(i)}\rangle \geq 0\}$. Let $i = i + 1$ and go to Step 1.
```

The complexity analysis of the above ellipsoid algorithm is based on the following lemma.

Lemma 82.4 [25]

For any convex set $A \subseteq \mathbb{R}^n$ and any vector $a \in \mathbb{R}^n \setminus \{0\}$, define A' by

$$A' = \{y \in A \mid \langle a, y - x_0\rangle \geq 0\}$$

where x_0 is the center of a γ-maximal ellipsoid of A. Then,

$$\mu(A') \leq 0.843\gamma^{-2}\mu(A)$$

Theorem 82.3

The ellipsoid algorithm will terminate within $O(n(E_1 + E_2))$ iterations.

Proof

Assume that the algorithm does not reach a strong ϵ-approximate equilibrium until the $(k+1)$-th step. For any $1 \le i \le k$, let H_i be the separate hyperplane, i.e., $H_i = \{x \in \mathbb{R}^n \mid \langle x - p^{(i)}, Z(p^{(i)}) \rangle = 0\}$. Since $p^{(i)}$ is not a strong ϵ-approximate equilibrium, we have

$$\langle Z(p^{(i)}), \bar{p} - p^{(i)} \rangle \ge \delta$$

by Lemma 82.3, where $\bar{p}$ is an equilibrium price. This implies that

$$dist(\bar{p}, H_i) \ge \langle \bar{p} - p^{(i)}, Z(p^{(i)}) \rangle / \|Z(p^{(i)})\|_2 \ge \delta/2^{E_2} \tag{82.23}$$

Let $\delta' = \delta/2^{E_2}$ and $B_{\delta'}(\bar{p})$ denote the ball centered at $\bar{p}$ with radius δ'. The inequality (82.23) implies that the algorithm cannot cut off any point in $B_{\delta'}(\bar{p})$ before the $(k+1)$-th step. By Lemma 82.4, the algorithm decreases $\mu(A_i)$ by a factor of $0.843\gamma^{-2}$. Then,

$$\mu(A_k) \le (0.843\gamma^{-2})^k \mu(A_0) = 0.861^k \mu(A_0)$$

Note that $B_{\delta'}(\bar{p}) \subseteq A(k)$, we have

$$\frac{(\delta')^n}{0.5^n} \le \frac{\mu(B_{\delta'}(\bar{p}))}{\mu(A_0)} \le \frac{\mu(A_k)}{\mu(A_0)} \le 0.861^k.$$

This implies that $k \le 4.64n(\log(\frac{1}{\delta}) + E_2 - 1) = O(n(E_1 + E_2))$ □

Now we return to the proof of Lemma 82.3 to conclude this section.

Proof of Lemma 82.3

Without loss of generality, we may assume that $\bar{p} = (1, 1, \ldots, 1)^T$ is an equilibrium price by scaling the units of goods. After scaling the units, the domain of demand functions turns to be $\Delta^+ = \{p \in \mathbb{R}^n_+ \mid 2^{-L_1} \le p_j \le 2^{L_1}\}$, where L_1 is polynomial of the input size. Assumption 2 turns to be $|\frac{\partial Z_i}{\partial p_j}(p)| \le 2^{D_1}$ for any i, j and any $p \in \Delta^+$, where D_1 is bounded by a polynomial in the input size.

Let $W_j = \sum_{i=1}^m w_{ij}$. Assume that the price $p = (p_1, p_2, \ldots, p_n)^T$ satisfies $p_1 \le p_2 \le \cdots \le p_n$, otherwise we can change the order of goods. We need to prove that $\sum_{j=1}^n Z_j(p) \ge \delta$.

Define a sequence of prices $\{\pi^s \mid s = 1, 2, \ldots, n\}$ as follows:

$$\pi_j^s = \begin{cases} p_j, & \text{when} \quad 1 \le j \le s-1 \\ p_s, & \text{when} \quad s \le j \le n \end{cases}$$

Note that $\pi^1 = (p_1, p_1, \ldots, p_1)^T$ is an equilibrium price. Using WGS, we can assert that

$$Z_j(\pi^{s+1}) \ge Z_j(\pi^s), \qquad \text{when} \quad 1 \le j \le s \tag{82.24}$$

$$Z_j(\pi^{s+1}) \le Z_j(\pi^s) \le 0, \qquad \text{when} \quad s+1 \le j \le n \tag{82.25}$$

Moreover, Arrow et al. [22] prove that $\sum_{j=1}^n (Z_j(\pi^{s+1}) - Z_j(\pi^s)) \ge 0$ for any $1 \le s \le n-1$. If we can further show that $\sum_{j=1}^n (Z_j(\pi^{s+1}) - Z_j(\pi^s)) \ge \delta$ for some s, the lemma is proved.

For any $1 \le s \le n-1$, we have

$$\sum_{j=1}^n \pi_j^{s+1} Z_j(\pi^{s+1}) - \sum_{j=1}^n \pi_j^s Z_j(\pi^s) = 0$$

by the Walras law. This implies

$$p_{s+1}\sum_{j=1}^{m}\big(Z_j(\pi^{s+1})-Z_j(\pi^{s})\big)=\sum_{j=1}^{s}(p_{s+1}-p_j)\big(Z_j(\pi^{s+1})-Z_j(\pi^{s})\big)-\sum_{j=s+1}^{m}(p_{s+1}-p_s)Z_j(\pi^{s})$$

$$\text{(by Eq.(82.24) and Eq.(82.25))}\geq\sum_{j=1}^{s}(p_{s+1}-p_s)\big(Z_j(\pi^{s+1})-Z_j(\pi^{s})\big)$$

Therefore,

$$\sum_{j=1}^{n}\big(Z_j(\pi^{s+1})-Z_j(\pi^{s})\big)\geq\frac{p_{s+1}-p_s}{p_{s+1}}\sum_{j=1}^{s}\big(Z_j(\pi^{s+1})-Z_j(\pi^{s})\big)\tag{82.26}$$

for any $1\leq s\leq n-1$.

Since $\pi^n = p$ is not a strong ϵ-approximate equilibrium, we have $Z_l(\pi^n)\geq\epsilon W_l$ for some good l. Note that $Z_l(\pi^1)\leq 0$ since π_1 is an equilibrium price. Hence, there must exists a k such that $Z_l(\pi^{k+1})-Z_l(\pi^k)\geq(\epsilon/n)W_l$. By Assumption 2, the significant change of the aggregate excess demand function is caused by the significant change of the price, that is $p_{k+1}-p_k\geq\epsilon W_l/n^2 2^{D_1}$.

By Eq. (82.26), we have

$$\begin{aligned}\sum_{j=1}^{n}\big(Z_j(\pi^{k+1})-Z_j(\pi^{k})\big)&\geq\frac{p_{k+1}-p_k}{p_{k+1}}\sum_{j=1}^{k}\big(Z_j(\pi^{k+1})-Z_j(\pi^{k})\big)\\&\geq\frac{\epsilon W_l}{n^2 2^{D_1}2^{L_1}}\big(Z_l(\pi^{k+1})-Z_l(\pi^{k})\big)\\&\geq\frac{\epsilon W_l}{n^2 2^{D_1}2^{L_1}}\frac{\epsilon W_l}{n}=\delta\end{aligned}$$

This completes the proof.

82.5 Approximation Algorithm for Indivisible Goods

In the last two sections, the convex programming methods and ellipsoid algorithm can compute an ϵ-approximate equilibrium in polynomial time with respect to $\log(1/\epsilon)$ and the input size. Thus, they can be viewed as polynomial-time algorithm for computing exact solutions, since their running time is polynomial whenever the exact solution is rational and can be encoded in polynomial length with respect to the input size (e.g., when the utility functions are linear or log-linear).

Approximation algorithms are necessary in several situations. There are cases where general equilibrium price does not exist, such as for some utility functions that are not concave, as well as for indivisible goods. In addition, even for cases the existence is guaranteed, computation of the equilibrium price is not known to be in polynomial time. For example, the equilibrium price may be an irrational number.

In this section, we should present some techniques that are able to find approximate solutions in polynomial time for some special cases.

For an exchange economy of linear utility functions with a bounded number of indivisible goods, Deng et al. [11] proved that there exists a polynomial time algorithm for finding an approximate solution, if an exact equilibrium price exists.

Theorem 82.4 **(Deng [11])**

If the number of goods is bounded, there is a polynomial-time algorithm which, for any linear indivisible market for which a price equilibrium exists, and for any $\epsilon > 0$, finds an ϵ-approximate equilibrium.

We shall present the algorithm and the proof for $n = 2$ and fixed ϵ.

One basic observation is that in any exact equilibrium price vector p, because of the market clearance condition, every agent exhausts her budget, that is, if x_i is the acquired vector of goods, $\langle p, x_i\rangle = \langle p, w_i\rangle$, or $\langle p, x_i - w_i \rangle = 0$. That is, the price vector is perpendicular to the exchange vector of goods for every agent.

The algorithm tries to find or approximate the equilibrium price vector $p = (p_1, p_2)$ where p_1, p_2 are, without loss of generality, integers between 0 and M (or, normalized for $p_1 \neq 0$ as $(1, q)$ with q between $1/M$ and M), where M is the sum of all total allocations of goods. Once we have p, we can always find the optimal allocations for each agent—this is an integer program with bounded variables [26,27].

In the assumed equilibrium if every agent has an allocation with all components small integers (say, less than $\frac{1}{\delta}$), then we can find the prices in this equilibrium (and therefore approximate the allocations via a method combining geometric rounding and dynamic programming) exhaustively in polynomial time.

We can therefore assume that we are looking for prices and allocations such that at least one agent has at least one large component. Define the set of integers $\tilde{Z}_+ = \{0, 1, \ldots, \lceil 1/\delta \rceil\} \cup \{\lceil (1+\delta)^i \rceil : i \geq 0, (1+\delta)^i < M\}$. That is, $\tilde{Z}_+$ contains all small numbers, plus all rounded powers of $(1+\delta)$ up to M. For an integer x, $\tilde{x}$ will denote the largest integer in $\tilde{Z}_+ \leq x$.

Define now the set of all price vectors p such that $\left\langle p^t, u \right\rangle = \left\langle p^t, w_i \right\rangle$ for some $u \in \tilde{Z}_+$ and some i. Notice that this set is polynomially large. We call them type 1 price vectors. We further add two more classes of price vectors by expanding the above set in two ways. First, for each w_k and each small points (i, j) with $i, j \leq 1/\delta$, we add a price vectors p that makes $\left\langle p^t, w_k \right\rangle < \left\langle p^t, (i, j) \right\rangle$ "barely hold." That is, (i, j) is not a feasible solution for agent k under the price vector p, and there is no integer point between the line $\left\langle p^t, x \right\rangle = \left\langle p^t, w_k \right\rangle$ and the line defined by w_k to (i, j). We call these the type 2 price vector. We also consider price vectors of type 3, i.e., those that are parallel to the utility functions of agents. The resulting set of price vectors is denoted by $\mathcal{V}$.

The algorithm is the following:

```
repeat for all price vectors $p \in \mathcal{V}$ (note: polynomially many):
  repeat for each agent $i$:
    find (by integer programming) the optimum utility $U_i$;
    find the set $A_i = \{a \in \tilde{Z}_+^2 : u_i(a) \geq (1-\epsilon)U_i, \left\langle p^t, a \right\rangle \leq \left\langle p^t, w_i \right\rangle\}$;
    find (by dynamic programming) $a_i \in A_i$, $i = 1, \ldots, m$ such that $\sum(1-\epsilon)\sum w_i \leq q \sum a_i \leq \sum w_i$.
  choose the $p$ for which such allocation exists.
```

Lemma 82.5

If there is an indivisible equilibrium, the above algorithm, finds an ϵ-equilibrium.

Among all price vectors in $\mathcal{V}$, we choose one that is the closest (in the angle) to the integer equilibrium price vector p^* and denote it by p^o. Let v_i^o be the optimal utility function value of agent i under price vector p^o, and let a_i^* be the optimal allocation to agent i that clears the market.

The main idea of the proof is that, under the price vector p^o, either a_i^* is a feasible solution to agent i; or not all the coordinates of a_i^* are small and rounding its large values to the larger $\lfloor (1+\delta)^j \rfloor$ smaller than a_i^* is a feasible solution to agent i. Moveover, in both cases, the utility value of agent i at such points are close to its optimal utility function value.

Consider the normalized price vector: $p^o = (1, p_2^o)$ and $p^* = (1, p_2^*)$. Without loss of generality, assume that a_i^* is to the northwest of w_i (i.e., the x-coordinate of a_i^* is smaller than that of w_i, y-coordinate larger). We need to consider two cases: $p_2^o \leq p_2^*$ and $p_2^o > p_2^*$.

If $p_2^o \leq p_2^*$, a_i^* is a feasible solution for agent i under price vector p^o. We should show that it is a good approximate solution for agent i under the approximate price vector. Let a_i^o denote the optimal solution of agent i under price vector p^o. Then $v_i^o = \left\langle p^o, a_i^o \right\rangle \geq \left\langle p^o, a_i^* \right\rangle$. Consider the ray r_i^o from w_i to a_i^o and the ray r_i^* from w_i to a_i^*; there is no point in $\tilde{Z}_+$ that is contained in the sector from r_i^o to r_i^* in the counter-clockwise order, by the choice of p^o as the closest to p^* among the three types of vectors (in particular, type 1). Therefore, the largest point f in $\tilde{Z}_+^2$ that is closest to a_i^o is a feasible point for agent i under price vector p^*. The utility function of agent i at point f is no more than that at point a_i^*. On the other hand, a_i^o is not a small point, i.e., one of its coordinate is large, because of type 2 price vectors. Therefore, the utility function value of agent i at point f is close to that at a_i^o (at least a factor of $1/(\delta + 1)$

of that at a_i^o). It follows that the utility function value of agent i at point a_i^* is close to that at a_i^o (within a factor of $1/(\delta + 1)$). For the second case, let $p_2^o > p_2^*$. The point a_i^* may not be feasible under price vector p^o. Because of type 3 price vectors, a_i^o is still to the northwest of w_i, the initial endowment to agent i. (Otherwise, if a_i^o is to the southeast of w_i, p^* will have to be parallel to the utility function of agent i and p^o will be the same as p^* by our choice.) Since a_i^o is feasible under p^*, the utility function value of agent i at it is no larger than that at a_i^*. Note that a_i^* is not a small point. Therefore, the largest point in $\tilde{Z}_+^2$ that is smaller than a_i^* must be feasible under p^o (otherwise, p^o is not the closest to p^*). The utility function value of agent i at this point is close to that at a_i^* and therefore close to that at a_i^o (within a factor of $1/(1 + \delta)$).

Therefore, we have established that there is a point $b_i \in A_i$ such that $\sum(1-\epsilon)\sum w_i \leq \sum b_i \leq \sum w_i$. The next step is to find it using dynamic programming. Even though there are only a polynomial number of possible values for allocations to the agents, it is not immediate that the dynamic programming algorithm has a polynomial number of states, since summing up the allocations results in a super-polynomial number of states. Our solution is as follows: the allocated commodities are in units of $(1 + \delta)^j$, and we guess the maximum j such that one of the agent gets that in the above existential solution. Then, in the Dynamic Programming (DP), we change the values of commodities less than $(1+\delta)^{(j-C\log n)}$ to zero for a sufficiently large constant C. Therefore, we only have up to n^C entries in the DP and can thus solve in polynomial time. Once we get a solution (with revised values of commodities) satisfying the market constraints (and the above discussion guarantees there is one), we change them back to the original values. That could make the market constraints unsatisfied but only by no more than $n(1+\delta)^{j-C\log n}$, which is an arbitrarily small fraction with respect to $(1 + \delta)^j$. We then sacrifice the utility function of the agent with the maximum value to satisfy the market constraint. So far, we have focused on one commodity. We choose the other commodity by taking the minimum possible value. ■

This concludes the proof for the $n = 2$ case. Note that we only need to consider a polynomial number of price vectors for the approximate solution. Then a clever dynamic programming resolves the matter how to find the solution. To extend the idea established here to general constant dimension requires new ideas to select the representative price vectors. Other ideas would go through without much change. The details are left as an exercise for the readers. The main idea is to divide the price space into a polynomial number of polytopes such that prices in each polytope derives the same optimal allocation in the $\tilde{Z}_+^d$ for a d-dimensional space.

82.6 Hardness Results

In Section 82.3 and Section 82.4, we present two algorithms that can compute an ϵ-approximate equilibrium in polynomial time. Obviously, for the market models fit the convex programming methods, the set of equilibria is convex. For the market models fit the ellipsoid algorithm, the set equilibria is also convex due to the separation lemma. However, in many cases, the set of equilibria may be nonconvex or even disconnected. For example, the market with CES utility functions may admit multiple disconnected equilibria, when $\rho < -1$. It is an open problem that whether there exists an efficient algorithm to find at least one equilibrium point in a market that has multiple disconnected equilibria.

Recently, Codenotti et al. [28] proved that computing a market equilibrium in a market with Leontief utility functions is actually hard. Note that the Leontief utility function can be viewed as the limit of CES function when $\rho \to -\infty$. Their recent hardness results were developed through a one-to-one correspondence between the Nash equilibria in bi-matrix games and the market equilibria in *the pairing model* of Leontief economies. With the connection, hardness results in bi-matrix games [29] were carried over to the market equilibrium problem. Cases leading to NP-hardness include: (1) the uniqueness of the market equilibrium; (2) the existence of an equilibrium with positive prices on a given set of goods; and (3) the existence of an equilibrium with at least (or at most) k goods positive priced. Moreover, the one-to-one correspondence yields the #P-hardness of counting the number of equilibria in a Leontief economy, since counting the number of Nash equilibria in a bi-matrix game is already known to be #P-hard [30].

In this section, we present a direct proof of #P-hardness of counting the number of equilibria in a Leontief economy, without the aid of the connection to Nash equilibria.

A Leontief utility function u_i can be characterized by a vector $a^i \in \mathbb{R}^n_+$. The trader's utility on a bundle of goods $x^i \in \mathbb{R}^n_+$ is given by

$$u_i(x^i) = \min_{1\leq j\leq n} \left\{ x^i_j / a^i_j \mid a^i_j \neq 0 \right\}$$

In other words, a trader with a Leontief utility function demands a bundle of goods proportional to the vector a^i. We can define a *response function* d_i to present trader i's response with respect to a certain price p:

$$d_i(p) = \begin{cases} 0, & \text{if } \left\langle p, a^i \right\rangle = \left\langle p, e^i \right\rangle = 0; \\ +\infty, & \text{if } \left\langle p, a^i \right\rangle = 0, \left\langle p, e^i \right\rangle \neq 0; \\ \frac{\left\langle p, e^i \right\rangle}{\left\langle p, a^i \right\rangle}, & \text{otherwise.} \end{cases}$$

Therefore, given a price p, trader i's demand to good j is $d_i(p)a^i_j$. Obviously, $u_i(d_i(p)a^i) = d_i(p)$. So $d_i(p)$ can also be viewed as the trader i's optimal utility under the price p.

We adopt (A, E) to denote a Leontief economy, where A is an $n \times m$ matrix whose ith column presents trader i's demands, and E is an $n \times m$ matrix whose ith column is trader i's endowments.

With these notations, the equilibrium in a Leontief economy is a pair of vectors (p, w) satisfying the following conditions:

$$\begin{array}{ll} w_i = d_i(p), & p \geq 0,\ p \neq 0 \\ Aw \leq \mathbf{1}, & p^T(Aw - \mathbf{1}) = 0 \end{array}$$

In this section, we will prove that counting the number of equilibria in a Leontief economy is $\#P$-hard by a reduction from #*SAT*.

Let ϕ be a Boolean formula in 3-conjunctive normal form. Let $V = \{x_1, x_2, \ldots, x_n\}$ be its set of variables, $L = \left\{l^0_1, l^1_1, l^0_2, l^1_2, \ldots, l^0_n, l^1_n\right\}$ the set of corresponding literals, and $C = \{c_1, c_2, \ldots, c_m\}$ its set of clauses.

We will construct a Leontief economy (A, E) for ϕ. Let B denote the set of traders and G denote the set of goods. There are $|V| + |L| + 1 = 3n + 1$ traders in the economy, denoted by

$$B = \{x_1, x_2, \ldots, x_n\} \cup \left\{l^0_1, l^1_1, l^0_2, l^1_2, \ldots, l^0_n, l^1_n\right\} \cup \{\phi\}$$

There are $2|V| + |L| + |C| = 4n + m$ goods in the economy. We also denote them by

$$G = \{x_1, x_2, \ldots, x_n\} \cup \left\{l^0_1, l^1_1, l^0_2, l^1_2, \ldots, l^0_n, l^1_n\right\} \cup \{c_1, \ldots, c_m\} \cup \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$$

The ambiguity of using the same symbols for both the trades and the goods should be clear from the context.

Let $A(b, g)$ denote trader b's demand for good g in proportion and $E(b, g)$ denote trader b's initial endowment of good g. Then the economy (A, E) is defined as follows:

$$A(b, g) = \begin{cases} 1, & \text{if } b = x_i, g = l^j_i, 1 \leq i \leq n,\ j = 0, 1; \\ 4/5, & \text{if } b = x_i, g = \sigma_i, 1 \leq i \leq n; \\ 1, & \text{if } b = l^j_i, g = x_i, 1 \leq i \leq n,\ j = 0, 1; \\ 1, & \text{if } b = g = l^j_i, 1 \leq i \leq n,\ j = 0, 1; \\ 1, & \text{if } b = l^j_i, g = c_k, l^j_i \in c_k, \forall i,\ j,\ k; \\ 2, & \text{if } b = l^j_i, g = c_k, l^j_i \notin c_k \text{ and } l^{1-j}_i \in c_k, \forall i,\ j,\ k; \\ 0, & \text{otherwise.} \end{cases}$$

$$E(b, g) = \begin{cases} 1, & \text{if } b = g = l^j_i, \forall i,\ j; \\ 1/n, & \text{if } b \in V, g \in V \cup L; \\ 5, & \text{if } b = \phi, g \in C; \\ 1, & \text{if } b = \phi, g = \sigma_i, 1 \leq i \leq n; \\ 0, & \text{otherwise.} \end{cases}$$

In the economy (A, E), the total amount of good g is 1 (for $g \in V$ or $g = \sigma_i$, $1 \le i \le n$), 2 (for $g \in L$) or 5 (for $g \in C$).

First, we will show that any truthful assignment of ϕ always corresponds an equilibrium in the economy. Assume there is a truth assignment $(l_1, \ldots, l_n)$ satisfies ϕ. Set $P(g) = 1$ for all $g = l_i^j = l_i$ and $P(g) = 0$ otherwise. If $b = \phi$ or $b = l_i^j \neq l_i$, the initial endowment of b would be valued as zero. By our convention of defining the response function, those agents will not want anything and their utility values will be zero.

If $b = x_i$, then $E(b, g) = 1/n$ for all $g \in V \cup L$. It follows that the value of the initial endowment of b is $\frac{1}{n} \times n = 1$ as $P(g) = 1$ for $g = l_i^j = l_i$ and $P(g) = 0$ otherwise. If $b = l_i^j = l_i$, then the value of the initial endowment of b is also 1 as both $E(b, g) = 1$ and $P(g) = 1$ for $b = g = l_i^j = l_i$.

By the definition of Leontief utility functions, we derive the utilities $U(b)$ of agent b as follows:

$$U(b) = \begin{cases} 1, & \text{when} \quad b = x_i, \forall 1 \le i \le n; \\ 1, & \text{when} \quad b = l_i^j = l_i, \ \forall 1 \le i \le n; \\ 0, & \text{when} \quad b = l_i^j \neq l_i, \ \forall 1 \le i \le n; \\ 0, & \text{when} \quad b = \phi \end{cases}$$

For the Leontief economy, the trader b's demand for good g is $A(b, g)U(b)$. Therefore, $\sum_b A(b, g)U(b)$ is the total amount of consumed good g. And $\sum_g A(b, g)P(g)U(b)$ is the total amount of money spent by trader b. The equilibrium conditions are:

$$1. \sum_g A(b, g)P(g)U(b) \le \sum_g E(b, g)P(g), \forall b \in B$$

$$2. \sum_b A(b, g)U(b) \le \sum_b E(b, g), \forall g \in G$$

$$3. \sum_g P(g)\left(\sum_b A(b, g)U(b) - \sum_b E(b, g)\right) = 0$$

For the first set of inequalities, we only need to consider those traders with endowment of nonzero value under the price vector P defined above.

1. For agent b: $b = x_i$, it desires two types of goods l_i^0 and l_i^1, of which one's price is zero and another's price is one. Therefore, with one unit of wealth, it will acquire one unit each of l_i^0 and l_i^1.
2. For agent b: $b = l_i^j = l_i$, it desires goods $g = x_i$ and $g = l_i^j$, $g = c_k$ such that $l_i^j \in c_k$, as well as $g = c_k$ such that $l_i^j \notin c_k$ but $l_i^{1-j} \in c_k$. All those goods cost zero except $g = l_i^j$ cost one per unit. Therefore, $b = l_i^j$ will get one unit of $g = l_i^j$, one unit of x_i, one unit of c_k if $l_i^j \in c_k$, and two units of $g = c_k$ if $l_i^j \notin c_k$ but $l_i^{1-j} \in c_k$.

For the second set of inequalities, consider goods in $V \cup L$:

$$\sum_b A(b, g)U(b) = 1 = \sum_b E(b, g), \qquad \forall g \in V$$

$$\sum_b A(b, g)U(b) = 2 = \sum_b E(b, g), \qquad \text{for } g = l_i^j = l_i$$

$$\sum_b A(b, g)U(b) = 1 < \sum_b E(b, g), \qquad \text{for } g = l_i^j \neq l_i$$

For goods in C, one unit of $g = c_k$ is desired by each literal in c_k that is true, two units of $g = c_k$ is desired by each literal in c_k that is false. Since l_i's, $i = 1, 2, \ldots, n$, is a satisfying assignment, there is a true literal in each c_k of three literals. Therefore, at most five units of $g = c_k$ are desired. With five units of each type of such goods, we have enough for them.

Finally, as the price of $g = l_i^{1-j}$ with $l_i = l_i^j$ is zero, and the prices of all c_k's are all zero, we have

$$\sum_g P(g)\left(\sum_b A(b, g)U(b) - \sum_b E(b, g)\right) = 0$$

On the other hand, we need to show that any equilibrium in the economy must correspond to a truthful assignment of ϕ. We declare that for any equilibrium (P, U) and any $i \in \{1, 2, \ldots, n\}$, either $U(x_i) \geq 3/2$, or $U(x_i) = U(l_i^j) = 1$ and $U(l_i^{1-j}) = 0$ for some $j \in \{0, 1\}$.

To see that, if both $P(l_i^j) \neq 0$ and $P(l_i^{1-j}) \neq 0$, then $U(l_i^j) + U(x_i) = 2$ and $U(l_i^{1-j}) + U(x_i) = 2$ as shown above. Therefore, $U(l_i^j) = U(l_i^{1-j})$. On the other hand, $U(l_i^0) + U(l_i^1) \leq 1$, also shown above. It follows that $U(x_i) \geq 3/2$ in this case.

Alternatively, assume that $P(l_i^j) \neq 0$ but $P(l_i^{1-j}) = 0$. Then, $U(l_i^j) + U(x_i) = 2$ by the former. Because of the latter, trader $b = l_i^{1-j}$ has a zero initial wealth and $U(l_i^{1-j}) = 0$. Therefore, the one unit of x_i must be all sold to $b = l_i^j$ since it is desired only by $b = l_i^j$ and $b = l_i^{1-j}$. Hence, $U(l_i^j) = 1$. In this case, $U(x_i) = 1$.

However, the goods σ_i forces that $U(x_i) \leq 5/4$. Hence, in the economy (A, E), there is a one-to-one correspondence between its equilibria and satisfiable truth assignments of the Boolean formula ϕ. This correspondence proves the $\#P$-hardness of counting the number of equilibria in a Leontief economy:

Theorem 82.5

Counting the number of equilibria in a Leontief economy is $\#P$-hard.

82.7 Concluding Remarks

The understanding of computational complexity issues for general equilibrium computation has been a recent focus that attracted scientists from different disciplines, including computer science, economics, and operations research. In addition to the related technical algorithmic questions for the mathematical problem, new concepts have emerged to fully characterize the proper definition of approximation for the ideal general equilibrium.

Much progress has been made in this field. Still several important open problems remains. The most important of all, what is the computational complexity for finding the equilibrium price, as guaranteed by the Arrow–Debreu theorem. Second, how to handle the dynamic case is especially interesting in theory, mathematical modelling, and algorithmic complexity as bounded rationality. Great progress must be made in those directions for any theoretical work to be meaningful in practice. Third, incentive compatible mechanism design protocols for the auction models have been most actively studied recently, especially with the rise of e-commerce. As an example, it has been successfully applied to study a dynamic model of auction [31]. Especially at this level, a proper approximate version of the equilibrium concept handling price dynamics should be especially important. Finally, there is a potential for more interplays between microeconomics and macroeconomics. It is possible that microeconomics methodology and data could be playing a bigger role in macroeconomics. Computational efficiency will without question be an important and necessary precondition. The conceptual approximation and the computational approximation also require a uniform framework.

Our discussion shows that significant progress has been made in the above directions but only as a first step. New ideas and methods have already been invented and applied in reality. The next significant step will soon manifest itself with many active studies in microeconomic behavior analysis for E-commercial markets. Nevertheless, the algorithmic analytic foundation we have laid down here will be an indispensable tool for further development in this reincarnated exciting field.

References

[1] Smith, A., *An Inquiry into the Nature and Causes of the Wealth of Nations*, Edinburgh, 1776. Reprinted by The Adam Smith Institute, London, 2001.

[2] Walras, L., *Elements of Pure Economics, or the Theory of Social Wealth*, 1874.

[3] Arrow, K. J. and Debreu, G., Existence of an equilibrium for a competitive economy, *Econometrica*, 22(3), 265, 1954.

[4] Scarf, H., The approximation of fixed points of a continuous mapping, *SIAM J. Appl. Math.*, 15, 1328, 1967.

[5] Bergman, L., Jorgenson, D. W., and Zalai, E., Eds., *General Equilibrium Modeling and Economic Policy Analysis*, Basil Blackwell, Cambridge, United Kingdom, 1990.

[6] Dervis, K., de Melo, J., and Robinson, S., *General Equilibrium Models for Development Policy*, World Bank Research Publication, Cambridge University Press, Cambridge, 1982.

[7] Francois, J. F. and Reinert, K., *Applied Methods for Trade Policy Analysis: A Handbook*, Cambridge University Press, Cambridge, 1997.

[8] Hirsch, M., Papadimitriou, C., and Vavasis, S., Exponential lower bounds for finding Brouwer fixed points, *J. Complex.*, 5, 379, 1989.

[9] Chen, X. and Deng X., Matching algorithmic bounds for finding Brouwer fixed point, *Proc. STOC,* 2005, p. 323.

[10] Allen, B., Approximate equilibria in microeconomic rational expectation models, *J. Econ. Theory*, 26, 244, 1982.

[11] Deng, X., Papadimitriou, C., and Safra, S., On the complexity of price equilibria, *JCSS*, 67(2), 311, 2002.

[12] Simon, H. A., Theories of bounded rationality, in *Decision and Organization: A Volume in Honor of Jacob Marschak*, McGuire, C. B. and Radner, R., eds., North-Holland Publishing Company, Amsterdam, London, 1972, chap. 8.

[13] Devanur, N. R. and Vazirani, V. V., An improved approximation scheme for computing Arrow–Debreu prices for the linear case, in *Lecture Notes in Computer Science*, Vol. 2914, Springer, Berlin, 2003, p. 149.

[14] Jain, K., Mahdian, M., and Saberi, A., Approximating market equilibria, in *Lecture Notes in Computer Science*, Vol. 2764, Springer, Berlin, 2003, p. 98.

[15] Angelopoulos, S., Sarma, A. D., Magen, A., and Viglas, A., On-line algorithms for market equilibria, in *Lecture Notes in Computer Science*, Vol. 3595, Springer, Berlin, 2005, p. 596.

[16] Nenakhov, E. and Primak, M., About one algorithm for finding the solution of the Arrow–Debreu model, *Kibernetica*, 3, 127, 1983.

[17] Jain, K., A polynomial time algorithm for computing the Arrow–Debreu market equilibrium for linear utilities, *Proc. FOCS,* 2004, p. 286.

[18] Jain, L., Vazirani, V. V., and Ye, Y., Market equilibria for homothetic, quasi-concave utilities and economies of scale in production, *Proc. SODA,* 2005, p. 63.

[19] Ye, Y., A path to the Arrow–Debreu competitive market equilibrium, *Math. Prog.*, (in press). Published online: 15 December 2006.

[20] Eisenberg, E. and Gale, D., Consensus of subjective probabilities: the pari-mutuel method, *Annal. Math. Stat.*, 30, 165, 1959.

[21] Eisenberg, E., Aggregation of utility functions, *Manage. Sci.*, 7(4), 337, 1961.

[22] Arrow, K. J., Block, H. D., and Hurwicz, L., On the stability of the competitive equilibrium, II, *Econometrica*, 27(1), 82, 1959.

[23] Codenotti, B., Pemmaraju, S., and Varadarajan, K., On the polynomial time computation of equilibria for certain exchange economies, *Proc. SODA,* 2005, p. 72.

[24] Huang, Z., Khachiyan, L., and Sikorski, K., Approximating fixed-points of weakly contracting maps, *J. Complexity*, 15, 200, 1999.

[25] Tarasov, S. P., Khachiyan, L., and Erlikh, I. I., The method of inscribed ellipsoid, *Soviet Math. Dokl.*, 37, 226, 1988.

[26] Lenstra, A. K., Lenstra, H. W., Jr., and Lovasz, L., Factoring polynomials with rational coefficients, *Math. Ann.*, 261(4), 515, 1982.

[27] Lenstra, H. W., Jr., Integer programming with a fixed number of variables, *Math. Oper. Res.*, 8(4), 538, 1983.

[28] Codenotti, B., Saberi, A., Varadarajan, K., and Ye, Y., Leontief economies encode nonzero sum two-player games, *Proc. SODA*, 2006.

[29] Gilboa, I. and Zemel, E., Nash and correlated equilibria: some complexity considerations, *Games Econ. Behav.*, 1, 80, 1989.

[30] Deng, X. and Huang, L-S., On the complexity of market equilibria with maximum social welfare, *Inf. Proc. Lett.*, 97(1), 4, 2006.

[31] Chen, N., Deng, X., Sun, X., and Yao, A. C.-C., Dynamic price sequence and incentive compatibility (extended abstract). *Proc. ICALP*, 2004, p. 320.

[32] Chen, N., Deng, X., Sun, X., and Yao, A. C.-C., Fisher equilibrium price with a class of concave utility functions, *Proc. ESA*, *Lecture Notes in Computer Science*, Vol. 3221, 2004, p. 169.

83

Approximation Algorithms and Algorithm Mechanism Design

Xiang-Yang Li
Illinois Institute of Technology

Weizhao Wang
Illinois Institute of Technology

83.1 Introduction

The majority of the algorithms and protocols designed in computer science implicitly assumes that the participating computers/users will act as instructed—except, perhaps, for a few faulty or malicious ones. The Internet, which is composed of different *heterogeneous* and *autonomous* systems, raises a doubt about this common belief. Computing devices, owned by different people or organizations, will likely do what is most beneficial to their owners. For example, routing on the Internet today is as much about money as it is about traffic. The business relationships of an Internet Service Provider (ISP) largely dictate its routing policy. This leads to a number of well-known pathologies in today's routing mechanism.

A selfish user always finds the best strategy that maximizes his own gain when others' strategies are known. Through the interaction of selfish users, the system may reach a stable point (called *Nash equilibrium*) where no user can improve his gain by unilaterally switching his action. However, the outcome at a Nash equilibrium may be *inefficient*: it could be arbitrarily worse than the global optimum coordinated by all users together [1]. This is often called the *price of anarchy*, which happens everywhere, for example, the chaotic traffic in most developing countries on the crash of some electronic submission systems. In the latter example, majority of the authors submit their papers at the last minute because there are no incentives for them to submit earlier.

With the emergence of the network as the platform of computing (e.g., grid computing and peer-to-peer (P2P) networks), algorithms or protocols intended for selfish computers/users must be designed in advance to cope with selfishness since the outcome at Nash equilibria may be *inefficient*. Such protocols and algorithms will likely involve incentives (e.g., monetary payments) between or to/from selfish participants. Thus, the algorithms and protocols should take into account not only the computational issues, such as

CPU time, memory usage, communication cost, and robustness, but also the incentive issues, such as the selfishness of individual users, strategies used by users, and the possible cooperations among users. To ensure that the selfish behavior results in a desirable outcome, researchers proposed several different ways to deal with selfish behaviors. Previous approaches include detecting and then avoiding or punishing selfish agents, studying the worst-case inefficiency of a Nash equilibrium, influencing the selfish users' behaviors via a variety of ways, such as pricing policies, differentiated services, reputations, and algorithm mechanism design [2–13].

Among those approaches, Nisan and Ronen [14] proposed the framework of *algorithmic mechanism design* (AMD). The main idea of AMD is to find certain mechanism to affect the behavior of the selfish users such that every user maximizes its utility when it tells the truth no matter what other users do, which is known as *truthful* or *strategyproof.* Among those strategyproof mechanism, the most well known is the so-called Vickrey–Clarke–Groves (VCG) mechanisms [15–17]. The VCG mechanisms are applicable to mechanism design problems whose outputs optimize the *utilitarian* objective function, which is simply the sum of all agents' valuations. Nisan and Ronen applied VCG mechanisms to some fundamental problems in computer science, including shortest paths, minimum spanning trees, and scheduling on unrelated machines. Unfortunately, in practice, some objective functions are not utilitarian; even for those problems with a utilitarian objective function, sometimes it is impossible to find the optimal output in polynomial time unless P = NP. Someone may persist to use the VCG mechanism even when some approximation algorithms and heuristics are used to solve the problem. In the first part of this chapter, we will review several literatures [18–20], which show that it is almost universal that if we use some approximation algorithms and heuristics, then VCG mechanisms are no longer truthful.

In light of this failure, some mechanisms other than VCG mechanisms are needed to address these issues. Thus, in the second part of the chapter, we study how to design truthful mechanisms for *binary demand games* where the output of an agent is either "selected" or "not selected." Recall that a mechanism $M = (\mathcal{O}, \mathcal{P})$ consists of two parts, an output function $\mathcal{O}$ and a payment scheme $\mathcal{P}$. In contrast to the VCG mechanisms, the binary demand game does not require that the output should optimize the objective function. In fact, binary demand games do not even require the existence of an objective function. Given any output function $\mathcal{O}$ for a binary demand game, we show that a truthful mechanism $M = (\mathcal{O}, \mathcal{P})$ exists for the game if and only if $\mathcal{O}$ satisfies a certain monotonicity property (MP). We complement this existence theorem with a general framework to design such a payment scheme $\mathcal{P}$. Furthermore, we present general techniques to compute the payments when the output is a composition of the outputs of subgames through the operators "or" and "and"; through round-based combinations; or through intermediate results, which may be themselves computed from other subproblems.

We then further generalize the binary demand game to demand games in which the output of an agent could be characterized as any real number. We also present a necessary and sufficient condition for the existence of the truthful mechanism and illustrate how to find the truthful mechanism by two concrete examples. We conclude this chapter with some literature review and the discussion of some interesting future works.

83.2 Technical Preliminaries and Previous Works

83.2.1 Approximation Algorithms

A *computational optimization* problem is a problem to find a solution a in the feasible region Ω which has the minimum (or maximum) value of the objective function $g(a)$. Let $a_{opt} \in \Omega$ be the solution that minimizes or maximizes the objective function $g(a)$. A ρ*-approximation* algorithm $\mathcal{A}$, for $\rho \geq 1$, always finds the solution b such that $g(b) \leq \rho g(a_{opt})$ for a minimization problem and $g(b) \geq g(a_{opt})/\rho$ for a maximization problem. We denote ρ as the *approximation ratio* of the algorithm $\mathcal{A}$.

83.2.2 Algorithm Mechanism Design

A standard model for mechanism design is as follows. There are n agents $1, 2, \ldots, n$. Each agent i has some private information t_i, called its *type*, only known to itself. For example, the type t_i can be the cost that agent i incurs for forwarding a packet in a network or can be a payment that the agent is willing to pay for a good in an auction. The agents' types define the *type vector* $t = (t_1, t_2, \ldots, t_n)$. Each agent i has a set of strategies A_i from which it can choose. For each strategy vector $a = (a_1, \ldots, a_n)$, where agent i plays strategy $a_i \in A_i$, the *mechanism* $\mathcal{M} = (\mathcal{O}, \mathcal{P})$ computes an *output* $o = \mathcal{O}(a)$ and a *payment* vector $\mathcal{P}(a) = (\mathcal{P}_1(a), \ldots, \mathcal{P}_n(a))$. Here the payment $\mathcal{P}_i(\cdot)$ is the money given to agent i and depends on the strategies used by the agents.

A *valuation* function $v_i(t_i, o)$ assigns a monetary amount to agent i for each possible output o. For example, in an instance of unicast routing, the agents are the n nodes in the network. Agent i's type is its cost c_i of forwarding a (unit amount of) data packet. The space of feasible outputs consists of all paths that connect the source node and the destination node. The valuation of node k for a path connecting the source and the destination is $-c_k$ if node k is on the path, and 0 otherwise.

Let $u_i(t_i, o)$ denote the *utility* of agent i at the output o of the game, given its type t_i. Here, following a common assumption in the literature, we assume the utility for agent i is quasilinear, that is, $u_i(t_i, o) = v_i(t_i, o) + \mathcal{P}_i(a)$. Let $a_{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$ denote the strategies of all the other agents except i. Let (x, a_{-i}) denote the vector $(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_n)$. Sometimes, we also denote it as $a|^i x$. Usually, we let a_{-i} denote that agent i did not participate the game at all. We adopt the assumption in neoclassic economics that all agents are aiming to optimize their utilities.

A strategy a_i is called a *dominant* strategy for agent i if it maximizes agent i's utility for all possible strategies of the other agents, that is,

$$u_i(t_i, \mathcal{O}(a_i, b_{-i})) \geq u_i(t_i, \mathcal{O}(a_i', b_{-i}))$$

for all $a_i' \neq a_i$ and all strategies b_{-i} of the agents other than i.

A strategy vector $a^\star$ is called a *Nash equilibrium* if it maximizes the utility of each agent i when the strategies of all the other agents are fixed as $a^\star_{-i}$, that is,

$$u_i(t_i, \mathcal{O}(a^\star)) \geq u_i(t_i, \mathcal{O}(a_i', a^\star_{-i}))$$

for all i and all $a_i' \neq a_i^\star$. A system-wide goal in mechanism design is defined by a *an objective function* $g(\cdot)$, which selects the optimal output given the agents' types. Given a mechanism with output function $\mathcal{O}(\cdot)$, we say that the mechanism *implements* the objective function $g(\cdot)$ if the output optimizes the objective function for all possible agent types.

A *game* is defined as $\mathcal{G} = (\mathcal{S}, \mathcal{M})$, where $\mathcal{S}$ is the setting for the game $\mathcal{G}$. Here, $\mathcal{S}$ consists of the parameters of the game that are set before the game starts and do not depend on the players' strategies. As a concrete example, in a unicast routing game [14], the setting also includes the topology of the network, the source node, and the destination node. Throughout this chapter, unless explicitly mentioned otherwise, the setting $\mathcal{S}$ of the game is fixed and we focus on how to design $\mathcal{P}$ given an output $\mathcal{O}$. All information about a game $\mathcal{G}$, including the setting $\mathcal{S}$, the output function $\mathcal{O}$, and the payment scheme $\mathcal{P}$, is public knowledge except each agent i's actual type t_i, which is private information to agent i.

A *direct-revelation* mechanism is a mechanism in which the only actions available to each agent are to report its private type either truthfully or falsely to the mechanism. An *incentive compatible* (IC) mechanism is a direct-revelation mechanism in which if an agent reports its type t_i to the mechanism truthfully, then it will maximize its utility. Incentive compatibility captures the essence of designing a mechanism to overcome the self-interest of agents in that in an IC mechanism an agent will choose to report its private information truthfully, out of its own self-interest. A direct-revelation mechanism is *truthful* if reporting its truth type is a dominant strategy. In a direct-revelation truthful mechanism, the payment scheme should satisfy the property that, for each agent i,

$$v_i(t_i, \mathcal{O}(t)) + \mathcal{P}_i(t) \geq v_i(t_i, \mathcal{O}(t|^i t_i')) + \mathcal{P}_i(t|^i t_i')$$

A truthful mechanism wants each agent to report its private type truthfully by providing incentives to the agents. Another common requirement in the literature of mechanism design is the so-called *individual rationality* (IR): an agent's utility of participating in the output of the mechanism is not less than its utility of not participating.

Arguably the most important positive result in mechanism design is the generalized VCG mechanisms by Vickrey [17], Clarke [15], and Groves [16]. An objective function $g(o, t)$ is called *utilitarian* if it is $g(o, t) = \sum_i v_i(t_i, o)$. The VCG mechanisms apply to (affine) maximization problems where the objective function is utilitarian and the set of possible outputs is finite. A direct-revelation mechanism $\mathcal{M} = (\mathcal{O}, \mathcal{P})$ belongs to the VCG family if (1) the output $\mathcal{O}(t)$ computed based on the type vector t maximizes the utilitarian objective function, and (2) the payment to agent i is $\mathcal{P}_i(t) = \sum_{j \neq i} v_j(t_j, \mathcal{O}(t)) + h^i(t_{-i})$. Here $h^i(\cdot)$ is an arbitrary function of t_{-i}, which typically is $-\sum_{j \neq i} v_j(t_j, \mathcal{O}(t_{-i}))$ to guarantee the IR property. Green and Laffont [21] proved that, under mild assumptions, the VCG mechanisms are the only truthful mechanism for utilitarian maximization problems. VCG mechanisms can be further generalized to be applicable to problems with objective function $g(o, t) = \sum_i \beta_i v_i(t_i, o)$, where β_i $(1 \leq i \leq n)$ are fixed constants.

The output function of a VCG mechanism is required to maximize the utilitarian objective function. This makes the mechanism computationally intractable in many cases. Furthermore, replacing an optimal algorithm for computing the output with an approximation algorithm usually leads to untruthful mechanisms if a VCG payment scheme is used. In this chapter, we review the approaches to design a truthful mechanism that does not optimize a utilitarian objective function.

83.2.3 Binary Demand Games and General Demand Games

A *binary demand game* is a game $\mathcal{G} = (\mathcal{S}, \mathcal{M})$, where $\mathcal{M} = (\mathcal{O}, \mathcal{P})$ and the range of $\mathcal{O}$ is $\{0,1\}^n$. In other words, the output is an n-tuple vector $\mathcal{O}(t) = (\mathcal{O}_1(t), \mathcal{O}_2(t), \ldots, \mathcal{O}_n(t))$, where $\mathcal{O}_i(t) = 1$ (respectively, 0) means that agent i is (respectively, is not) selected in the output.

Hereafter, we make the following further assumptions:

1. The valuation of the agents are not *correlated*, that is, $v_i(t_i, o)$ is a function of t_i and o_i only and is denoted as $v_i(t_i, o_i)$.
2. The valuation $v_i(t_i, 0)$ is a publicly known value and is normalized to 0. This assumption is needed to guarantee the IR property.

Notice that in applications where agents provide services and receive payments, for example, unicast and job scheduling, the valuation v_i of an agent i is usually negative. For the convenience of presentation, we let $v_i(o_i) = v_i(t_i, o_i)$ and the *cost* of agent as $c_i = -v_i(t_i, 1)$, that is, it costs agent i c_i to provide the service. Throughout this chapter, we will use c_i instead of v_i in our analysis when v_i is negative. All our results can apply to the case where the agents receive the service rather than provide it by setting c_i to negative, as in auction. From now on, we will replace t_i with c_i.

In a binary demand game, if we want to optimize an objective function $g(o, \mathbf{c})$, then we call it a *binary optimization demand game*. The main differences between the binary demand games and those problems that can be solved by VCG mechanisms are as follows:

1. For the binary demand games, the objective function is arbitrary; for the VCG mechanisms, the objective function is utilitarian.
2. For the binary demand games, the output function $\mathcal{O}$ does not necessarily optimize an objective function; for the VCG mechanisms, the output function should optimize the objective function.
3. For the binary demand games, the range of the output function is $\{0,1\}^n$; for the VCG mechanisms, the range of the output function may be any finite set.
4. For the binary demand games, the agents' valuations are not correlated by assumption; for the VCG mechanisms, the agents' valuations may be correlated.

A *demand game* is a game $\mathcal{G} = (\mathcal{S}, \mathcal{M})$, where $\mathcal{M} = (\mathcal{O}, \mathcal{P})$ and the range of $\mathcal{O}$ is R^n. In other words, the output is an n-tuple vector $\mathcal{O}(\mathbf{c}) = (\mathcal{O}_1(\mathbf{c}), \ldots, \mathcal{O}_n(\mathbf{c}))$, where $\mathcal{O}_i(\mathbf{c})$ is a nonnegative real number and the valuation of agent i is $v_i(\mathbf{c}) = -c_i \cdot \mathcal{O}_i(\mathbf{c})$. In a demand game, we treat $\mathcal{O}_i(\mathbf{c})$ as the "*load*" on the agent i and c_i is the cost for a unit of load by agent i. The examples of demand games include but are not limited to

1. *Job scheduling.* Assign different jobs of different workload to different machines. Here, the machine is the agent and its cost is the unit load of the job.
2. *DiffServ multicast.* Given a set of receivers, each with its own quality (*e.g.*, bandwidth) requirements, find a multicast tree and the quality assignment for each link (or node) of the tree such that the receivers' requirements are satisfied. The links (or nodes) are agents and each of them has different costs to provide different qualities of service.

83.3 Limitations of VCG Mechanisms

83.3.1 Untruthfulness of VCG Mechanism: An Example

In this section, we begin with an example to show that VCG mechanism fails when it is simply coupled with almost all approximation algorithms. The example is the *set cover* problem: there is a set U of m elements needed to be covered, and each agent $1 \leq i \leq n$ can cover a subset of elements S_i with a cost c_i. Let $S = \{S_1, S_2, \ldots, S_n\}$ and $c = (c_1, c_2, \ldots, c_n)$. We want to find a subset of agents D such that $U \subseteq \bigcup_{i \in D} S_i$. The selected subsets S_i with $i \in D$ is called a *set cover* for U. The total cost $\sum_{i \in D} c_i$ is the objective function to be minimized. Clearly, this objective function is utilitarian and thus a VCG mechanism can be applied if we can find a D with the minimum cost. It is well known that finding the minimum cost set cover is NP-hard. In Ref. [22], an algorithm of approximation ratio of H_m has been proposed and it has been proved that this is the best ratio possible for the set cover problem, where $H_m = 1 + \frac{1}{2} + \cdots + \frac{1}{m} = O(\log\ m)$ is the harmonic number. For the completeness of our discussion, we review this algorithm as follows.

Let GSC$(\cdot)$ be the sets selected by Algorithm 83.1. Notice that the output set D is a function of S and c. Some works [23] assumed that the type of an agent is c_i, that is, S_i is assumed to be a public knowledge. Here, we consider a more general case in which the type of an agent is $\langle S_i, c_i \rangle$. In other words, we assume that every agent i can lie about not only its cost c_i but also the set S_i. The problem is now similar to the combinatorial auction with single-minded bidders in Ref. [24], with the following difference: in the set cover problem we want to cover all the elements and the chosen sets overlap while in combinatorial auction the chosen sets are disjoint.

Algorithm 83.1 Greedy Set Cover (GSC)

```
Input: A set $U$, $S = \{S_1, S_2, \ldots, S_n\}$, and $c = (c_1, c_2, \ldots, c_n)$.
Output: A set of agents $D$ that covers all the elements in $U$.
Initialize $r = 1$, $T_0 = \emptyset$, and $D = \emptyset$.
while $R \neq U$ do
    Find the set $S_j$ with the minimum density $\frac{c_j}{|S_j - T_r|}$.
    Set $T_{r+1} = T_r \bigcup S_j$ and add agent $j$ to the set $D$.
    $r = r + 1$
Output $D$.
```

Assume that we use Algorithm 83.1 to find a set cover, and want to apply VCG mechanisms to compute the payments to the selected agents. The payment to an agent i is 0 if $S_i \notin R$. Otherwise, its payment is

$$\mathcal{P}_i^{\mathrm{VCG}} = |\mathrm{GSC}(S \backslash S_i)| - |\mathrm{GSC}(S)| + c_i$$

Here $|T|$ is the total costs of the sets in T for $T \subseteq S$. Following theorem shows that the VCG payment $\mathcal{P}^{\mathrm{VCG}}$ is not truthful.

Theorem 83.1

The mechanism $M = (\text{GSC}, \mathcal{P}^{VCG})$ is not truthful, where GSC *is the set cover computed by Algorithm 83.1.*

Proof

We prove with the following counterexample. Let $U = \{x_1, \ldots, x_n\}$ and $S = \{S_1, \ldots, S_{n+1}\}$, where $S_i = \{x_i\}$ for $1 \leq i \leq n$, and $S_{n+1} = \{x_1, x_2, \ldots, x_n\}$. Let $c_i = \frac{1}{n-i+1}$, for $1 \leq i \leq n$, and $c_{n+1} = 1 + \epsilon$, where ϵ is a small positive number. If we apply Algorithm GSC, it is easy to show that the resulting output $\text{GSC}(S) = \{S_1, \ldots, S_n\}$ with $|\text{GSC}(S)| = \sum_{i=1}^{n} \frac{1}{i} = H_n$. We now consider what payment agent i will get. When we remove the set S_1, we can show that $\text{GSC}(S \backslash S_1) = S_{n+1}$, and consequently, $|\text{GSC}(S \backslash S_1)| = 1+\epsilon$. Thus, the payment to agent 1 is $p_1^{\text{VCG}} = 1 + \epsilon - H_n + 1/n$, which is less than its cost $1/n$. Thus, the mechanism $M = (\text{GSC}, \mathcal{P}^{\text{VCG}})$ is not truthful. □

83.3.2 Untruthfulness of VCG Mechanism: General Results

In this section, we show that the VCG mechanism not only fails for certain approximation algorithms, but also fails for almost all approximation algorithms satisfying certain weak properties. Notice that some algorithm may try to optimize some object function g that is not utilitarian. Thus, $\mu(o, t) = \sum_i v_i(t_i, o)$ is a utilitarian objective function while g could be an arbitrary objective function. Following is a definition that will be used later.

Definition 83.1

Let $\mathcal{A}$ be an algorithm that maps the declared type vector into allowable output. Let T^i be the allowable declared type for agent i, and $T = \Pi_{i=1}^{n} T^i$ be the space of all possible types. Let $\mathcal{O}^{\mathcal{A}}(T)$ be the range of the output from the space T. We say $\mathcal{A}$ is local maximal *in its range if for every pair of t and t′ in T such that they only differ in type t_j, $\mu(\mathcal{O}^{\mathcal{A}}(t), t) \geq \mu(\mathcal{O}^{\mathcal{A}}(t'), t)$.*

Following theorem characterizes the VCG mechanism with an approximation algorithm that is truthful.

Theorem 83.2

A VCG-based mechanism with an algorithm $\mathcal{A}$ is truthful if and only if $\mathcal{A}$ is local maximal in its range.

Proof

The sufficient condition follows directly from Definition 83.1 about the local maximal. We then prove that if a VCG-based mechanism with an algorithm $\mathcal{A}$ is truthful, then $\mathcal{A}$ is local maximal in its range. Recall that the utility for agent i is $\mu(\mathcal{O}^{\mathcal{A}}(t), t) + h_i(t_{-i})$. From the definition of IC, $\mu(\mathcal{O}^{\mathcal{A}}(t), t) + h_i(t_{-i}) \geq \mu(\mathcal{O}^{\mathcal{A}}(t|^i t_i'), t) + h_i(t_{-i})$. Thus, $\mu(\mathcal{O}^{\mathcal{A}}(t), t) \geq \mu(\mathcal{O}^{\mathcal{A}}(t|^i t_i'), t)$ for any agent i, which implies local maximal. □

It is not difficult to observe that that $\mathcal{A}$ is local maximal does not imply that it outputs the optimal solution. However, under most circumstance, we can show that local maximal is closely related to the optimal solution.

Theorem 83.3

If algorithm $\mathcal{A}$ is local maximal for the set cover game, then it is either optimal or has an arbitrary large approximation ratio.

Proof

Let t be a type vector such that $\mu(\mathcal{O}^{\mathcal{A}}(t), t)$ is not optimal over all possible outputs $o \in \mathcal{O}^{\mathcal{A}}(T)$. Then there exists an output $o^{opt} \in \mathcal{O}^{\mathcal{A}}(T)$ such that $\mu(o^{opt}, t)$ is maximal over all possible output $o \in \mathcal{O}^{\mathcal{A}}(T)$. Without loss of generality, we assume that $\mathcal{O}^{\mathcal{A}}(t^{opt}) = o^{opt}$. Recall for the set cover game, $\mathcal{O}_i^{\mathcal{A}}(t) = 0$ or 1 and the valuation of agent i is $v_i(o, t) = -o_i c_i$, where c_i is the cost of the set i. Thus, $\mu(o, t) = -\sum_{i=1}^{n} o_i c_i$. Since the type t_i is solely determined by c_i, we will abuse our notation and let $t_i = -c_i$. Therefore, $v_i(o, t) = o_i t_i$ and $\mu(o, t) = \sum_{i=1}^{n} o_i t_i$.

Let us consider a new type vector z such that $z_i = t_i^{opt}$ if $o_i^{opt} = 1$ and $z_i = \delta \sum_{j=1}^{n} t_j$ if $o_i^{opt} = 0$, where $\delta > 1$ is an positive number. Following we argue that $\mu(o^{opt}, z)$ is also maximal over all possible output $o \in \mathcal{O}^{\mathcal{A}}(T)$. Considering an $o \in \mathcal{O}^{\mathcal{A}}(T)$, if $o_i = 1$ and $o_i^{opt} = 0$, then $\mu(o, z) \leq o_i z_i = \delta \sum_{j=1}^{n} t_j < \mu(o^{opt}, z)$. Thus, we only need to consider the output $o \neq o^{opt}$ such that $o_i = 0$ if $o_i^{opt} = 0$. Since $o \neq o^{opt}$, there must exist an agent j such that $o_j = 0$ and $o_j^{opt} = 1$. Thus, $\mu(o, z) = \sum_{i=1}^{n} o_i z_i \leq \sum_{i=1}^{n} o_i t_i \leq \sum_{i=1}^{n} o_i^{opt} t_i = \sum_{i=1}^{n} o_i^{opt} z_i = \mu(o^{opt}, z)$, which is a contradiction. Thus, there does not exist such kind of output o, and in consequence, $\mu(o^{opt}, z)$ is also maximal over all possible output $o \in \mathcal{O}^{\mathcal{A}}(T)$.

Following we consider a series of type $t^{(i)}$ where $i = 0$ to n.

$$\begin{cases} \begin{aligned} t^{(0)} &= t = (t_1, t_2, \ldots, t_n) \\ t^{(1)} &= (z_1, t_2, \ldots, t_n) \\ &\vdots \\ t^{(i)} &= (z_1, z_2, \ldots, z_i, t_{i+1}, \ldots, t_n) \\ &\vdots \\ t^{(n-1)} &= (z_1, z_2, \ldots, z_{n-1}, \ldots, t_n) \\ t^{(n)} &= z = (z_1, z_2, \ldots, z_{n-1}, \ldots, z_n) \end{aligned} \end{cases}$$

Since $\mathcal{A}$ is local maximal, $\mu(\mathcal{O}^{\mathcal{A}}(t^{(i)}), t^{(i)}) \geq \mu(\mathcal{O}^{\mathcal{A}}(t^{(i+1)}), t^{(i)})$ for $0 \leq i \leq n-1$. In contrast, $t_j^{(i)} \geq z_j$ for every agent j. Thus, $\mu(\mathcal{O}^{\mathcal{A}}(t^{(i+1)}), t^{(i)}) \geq \mu(\mathcal{O}^{\mathcal{A}}(t^{(i+1)}), t^{(i+1)})$. Therefore, $\mu(\mathcal{O}^{\mathcal{A}}(t^{(i)}), t^{(i)}) \geq \mu(\mathcal{O}^{\mathcal{A}}(t^{(i+1)}), t^{(i+1)})$ for $0 \leq i \leq n-1$. This proves that $\mu(\mathcal{O}^{\mathcal{A}}(t), t) \geq \mu(\mathcal{O}^{\mathcal{A}}(z), z)$. Since $\mu(o^{opt}, z) = \mu(o^{opt}, t) > \mu(\mathcal{O}^{\mathcal{A}}(t), t)$, $\mathcal{O}^{\mathcal{A}}(z) \neq o^{opt}$. Thus, $\mu(\mathcal{O}^{\mathcal{A}}(z), z) \geq \delta \sum_{i=1}^{n} t_i \geq \delta \mu(o^{opt}, z)$. Notice that δ is arbitrary. Thus, the approximation ratio could be arbitrary large. This finishes our proof. □

Theorem 83.3 reveals a negative result for the set cover game and similar result holds for most of the game. Nisan and Ronen [18], proves that Theorem 83.3 also holds for a more general cost minimization game and a similar result holds for any approximation algorithm for the combinatorial auctions. One can further extend this idea to prove similar results for most approximation algorithms.

83.4 Approximation Algorithms and Binary Demand Game

We showed that the VCG mechanism is not truthful for most of the games if we use some approximation algorithm instead of the optimal solution. In observance of this negative result, we discuss how to design some non-VCG truthful mechanism for some commonly used approximation algorithms if possible. We discuss the truthful mechanism design for approximation algorithms in binary demand game in this section and general demand games in next section.

83.4.1 Characterize the Strategyproof Mechanism

In this section, we discuss how to characterize the strategyproof mechanism for a binary demand game. Given the output function $\mathcal{O}$ for a binary demand game, following is a sufficient and necessary condition for the existence of a truthful payment scheme $\mathcal{P}$. This condition has been observed by several literatures before, for more details refer to Refs. [19,23,25]. Before we present the main result, we review some lemmas proved in Ref. [19].

Lemma 83.1

If a direct revelation mechanism $\mathcal{M} = (\mathcal{O}, \mathcal{P})$ satisfies IC, then it satisfies the property that for any agent i, if $\mathcal{O}_i(t|^i t_{i_1}) = \mathcal{O}_i(t|^i t_{i_2})$, then $\mathcal{P}_i(t|^i t_{i_1}) = \mathcal{P}_i(t|^i t_{i_2})$.

Lemma 83.2

For any truthful mechanism for a binary demand game $\mathcal{G}$ with setting $\mathcal{S}$, if we fix the cost c_{-i} of all agents other than i, then the payment to agent i is a constant P_i^1 if $\mathcal{O}_i(c) = 1$, and it is another constant P_i^0 if $\mathcal{O}_i(c) = 0$.

Theorem 83.4

Fix the setting $\mathcal{S}$ for a binary demand game. If a mechanism $M = (\mathcal{O}, \mathcal{P})$ satisfies IC, then the mechanism $M' = (\mathcal{O}, \mathcal{P}')$ with the same output function $\mathcal{O}$ but with $\mathcal{P}_i'(c) = \mathcal{P}_i(c) - \delta_i(c_{-i})$ for any function $\delta_i(c_{-i})$ satisfies IC.

Definition 83.2 (MP)

An output function $\mathcal{O}$ for a game $\mathcal{G}$ is said to satisfy the MP *if for every agent i and two of its possible costs $c_{i_1} < c_{i_2}$, $\mathcal{O}_i(\mathbf{d}|^i c_{i_2}) \leq \mathcal{O}_i(\mathbf{d}|^i c_{i_1})$ for any* $\mathbf{d}$.

For a binary demand game, this definition implies that if $\mathcal{O}_i(c|^i c_{i_2}) = 1$, then $\mathcal{O}_i(c|^i c_{i_1}) = 1$. The following theorem is summarized in Ref. [19].

Theorem 83.5

Given an output function $\mathcal{O}$ for a binary demand game, the following three conditions are equivalent:

1. *There exists a value $\kappa_i(\mathcal{O}, c_{-i})$ (which we will call a cut value), such that $\mathcal{O}_i(c) = 1$ if $c_i < \kappa_i(\mathcal{O}, c_{-i})$ and $\mathcal{O}_i(c) = 0$ if $c_i > \kappa_i(\mathcal{O}, c_{-i})$.*
   Remark. *When $c_i = \kappa_i(\mathcal{O}, c_{-i})$, $\mathcal{O}_i(c)$ can be either* 0 *or* 1 *depending on the tie-breaker of the output function $\mathcal{O}$. Hereafter, we will not consider the tie-breaker scenario.*
2. *The output function $\mathcal{O}$ satisfies MP.*
3. *There exists a payment scheme $\mathcal{P}$ such that the mechanism $(\mathcal{O}, \mathcal{P})$ is truthful.*

We now formulate a general framework (Figure 83.1) for designing a truthful payment scheme for a binary demand game as follows.

83.4.2 Simple Composition Technique

In this section, we introduce techniques to compute the cut-value function by combining multiple output functions with conjunctions or disconjunctions. For simplicity, given an output function $\mathcal{O}$, we will use $\kappa(\mathcal{O}, c)$ to denote an n-tuple vector

$$(\kappa_1(\mathcal{O}, c_{-1}), \kappa_2(\mathcal{O}, c_{-2}), \ldots, \kappa_n(\mathcal{O}, c_{-n})),$$

General Framework 1: Design Payment for A Binary Demand Game Input: An output function $\mathcal{O}$ for a binary demand game.
Output: A payment scheme $\mathcal{P}$ such that the mechanism $(\mathcal{O}, \mathcal{P})$ is truthful.

GF-Stage 1: Check whether the output function $\mathcal{O}$ satisfies MP. If it does not, then there is no payment scheme $\mathcal{P}$ such that mechanism $M = (\mathcal{O}, \mathcal{P})$ is truthful. Otherwise, define the payment scheme $\mathcal{P}$ as follows.

GF-Stage 2: Based on the output function $\mathcal{O}$, find the cut value $\kappa_i(\mathcal{O}, c_{-i})$ for agent i such that $\mathcal{O}_i(c|^i d_i) = 1$ when $d_i < \kappa_i(\mathcal{O}, c_{-i})$, and $\mathcal{O}_i(c|^i d_i) = 0$ when $d_i > \kappa_i\ (\mathcal{O}, c_{-i})$.

GF-Stage 3: The payment for agent i is 0 if $\mathcal{O}_i(c) = 0$; the payment is $\kappa_i(\mathcal{O}, c_{-i})$ if $\mathcal{O}_i(c) = 1$.

FIGURE 83.1 General framework.

where $\kappa_i(\mathcal{O}, c_{-i})$ is the cut value for agent i when the output function is $\mathcal{O}$ and the costs c_{-i} of all other agents are fixed.

Theorem 83.6

Fix the setting $\mathcal{S}$ of a binary demand game. Assume that there are m output functions $\mathcal{O}^1, \ldots, \mathcal{O}^m$ satisfying MP, and $\kappa(\mathcal{O}^i, c)$ is the cut-value function vector for $\mathcal{O}^i$, where $i = 1, 2, \ldots, m$. Then the output function $\mathcal{O}(c) = \bigvee_{i=1}^{m} \mathcal{O}^i(c)$ also satisfies MP, and the cut-value function for $\mathcal{O}$ is

$$\kappa(\mathcal{O}, c) = \max_{i=1}^{m}\{\kappa(\mathcal{O}^i, c)\}$$

where $\kappa(\mathcal{O}, c) = \max_{i=1}^{m}\{\kappa(\mathcal{O}^i, c)\}$ denotes $\kappa_j(\mathcal{O}, c) = \max_i\{\kappa_j(\mathcal{O}^i, c_{-j})\}$ and $\mathcal{O}(c) = \bigvee_{i=1}^{m} \mathcal{O}^i(c)$ denotes $\mathcal{O}_j(c) = \mathcal{O}_j^1(c) \vee \mathcal{O}_j^2(c) \vee \cdots \vee \mathcal{O}_j^m(c)$ for every agent j, $1 \le j \le n$.

Proof

Assume that $c_i > c_i'$ and $O_i(c) = 1$. Without loss of generality, we assume that $O_i^k(c) = 1$ for some k, $1 \le k \le m$. From the assumption that $O_i^k(c)$ satisfies MP, we obtain that $O_i^k(c|^i c_i') = 1$. Thus, $\mathcal{O}_i(c|^i c_i') = \bigvee_{j=1}^{m} \mathcal{O}^j(c) = 1$. This proves that $\mathcal{O}(c)$ satisfies MP. The correctness of the cut-value function for $\mathcal{O}$ follows directly from Theorem 83.5. □

To demonstrate Theorem 83.6, we discuss a concrete example here.

In a network, sometimes we want to deliver a packet to a set of nodes instead of one. This problem is known as *multicast*. The most commonly used structure in multicast routing is the so called *shortest path tree* (SPT). Consider a network $G = (V, E, c)$, where V is the set of nodes, and vector c the actual cost of the nodes forwarding the data. Assume that the source node is s and the set of receivers is $Q \subset V$. For each receiver $q_i \in Q$, we compute the shortest path, that is, least cost path, denoted by $\mathsf{P}(s, q_i, d)$, from the source s to q_i under the reported cost profile d. The union of all such shortest paths forms the SPT.

We define $\text{LCP}^{(s,q_i)}$ as the output function corresponds to path $\mathsf{P}(s, q_i, d)$, that is, $\text{LCP}_k^{(s,q_i)}(d) = 1$ if and only if node v_k is in $\mathsf{P}(s, q_i, d)$. Then the output SPT is defined as $\bigvee_{q_i \in Q} \text{LCP}^{(s,q_i)}$. In other words, $\text{SPT}_k(d) = 1$ if and only if q_k is selected in some $\mathsf{P}(s, q_i, d)$. We now use General Framework 1 to design a payment scheme $\mathcal{P}$ such that the mechanism $\mathcal{M} = (\text{SPT}, \mathcal{P})$ is truthful. The output function LCP is a utilitarian and satisfies MP. Thus, from Theorem 83.6, SPT also satisfies MP, and the cut-value function vector for SPT is

$$\kappa(\text{SPT}, c) = \max_{q_i \in Q} \kappa(\text{LCP}^{(s,q_i)}, c)$$

where $\kappa(\text{LCP}^{(s,q_i)}, c)$ is the cut-value function vector for $\mathsf{P}(s, q_i, c)$. Consequently, the payment scheme $\mathcal{P}$ defined by General Framework 1 with respect to $\kappa(\text{SPT}, c)$ is the minimum among all payment Q such that (SPT, Q) is truthful.

The next theorem is a companion theorem of Theorem 83.6.

Theorem 83.7

Fix the setting $\mathcal{S}$ of a binary demand game. Assume that there are m output functions $\mathcal{O}^1, \ldots, \mathcal{O}^m$ satisfying MP, and $\kappa(\mathcal{O}^i, c)$ is the cut-value function for $\mathcal{O}^i$, where $i = 1, 2, \ldots, m$. Then the output function $\mathcal{O}(c) = \bigwedge_{i=1}^{m} \mathcal{O}^i(c)$ also satisfies MP. Moreover, the cut-value function for $\mathcal{O}$ is

$$\kappa(\mathcal{O}, c) = \min_{i=1}^{m}\{\kappa(\mathcal{O}^i, c)\}$$

The next corollary uses Theorems 83.6 and 83.7 to derive a new technique for computing a payment scheme for IF–THEN–ELSE statements.

Corollary 83.1

Assume there are two output function $\mathcal{O}^1$ and $\mathcal{O}^2$ satisfying MP, and $\kappa_i(\mathcal{O}^1, c_{-i})$ and $\kappa_i(\mathcal{O}^2, c_{-i})$ are the cut-value functions for $\mathcal{O}^1$, $\mathcal{O}^2$, respectively. Let cond(c) *be the function vector such that*

$$\mathrm{cond_i(c)} = \begin{cases} 0, & \kappa_i(\mathcal{O}^1, c_{-i}) - \delta_i^2(c_{-i}) \le c_i \le \kappa_i(\mathcal{O}^1, c_{-i}) + \delta_i^1(c_{-i}) \\ 1, & \textit{otherwise} \end{cases} \tag{83.1}$$

where $\delta_i^1(c_{-i})$ and $\delta_i^2(c_{-i})$ are two positive functions. Then the output $\mathcal{O}$ defined as

$$\mathcal{O} = \mathit{IF}(\mathrm{cond}(c))\ \mathit{THEN}\, \mathcal{O}^1\ \mathit{ELSE}\, \mathcal{O}^2$$

satisfy MP.

Proof

Notice that the function $\mathcal{O}_i(c)$ can be treated as $\mathcal{O}_i(c) = [(c_i \le \kappa_i(\mathcal{O}^1, c_{-i}) + \delta_i^1(c_{-i})) \wedge \mathcal{O}^2(c_{-i}, c_i)] \vee (c_i < \kappa_i(\mathcal{O}^1, c_{-i}) - \delta_i^2(c_{-i}))$. From Theorems 83.6 and 83.7, the output function $\mathcal{O}$ satisfies MP and $\kappa_i(\mathcal{O}, c_{-i}) = \max\{\min(\kappa_i(\mathcal{O}^1, c_{-i}) + \delta_i^1(c_{-i}), \kappa_i(\mathcal{O}^2, c_{-i})), \kappa_i(\mathcal{O}^1, c_{-i}) - \delta_i^2(c_{-i}))\}$. This finishes our proof. Observe that the output function $\mathcal{O}$ is exactly the IF–THEN–ELSE function in Ref. [23]. □

83.4.3 Complex Composition Technique

Some approximation algorithms are round-based, where each round of an algorithm selects some agents and updates the setting and the cost profile if necessary. For example, several approximation algorithms for minimum-weight vertex cover [26], minimum-weight set cover [22], and minimum-weight Steiner tree [27] fall into this category.

As an example, we discuss the *minimum-weighted vertex cover problem* (MWVC) [28] to show how to compute the cut values for a round-based output. Given a graph $G = (V, E)$, where the nodes $v_1, v_2, \ldots, v_n$ are the agents and each agent v_i has a weight c_i, we want to find a node set $V' \subseteq V$ such that for every edge $(u, v) \in E$ at least one of u and v is in V'. Such V' is called a *vertex cover* of G. The valuation of a node i is $-c_i$ if it is selected; otherwise its valuation is 0. For a subset of nodes $V' \in V$, we define its *weight* as $c(V') = \sum_{i\in V'} c_i$.

We want to find a vertex cover with the minimum weight. Hence, the objective function to be implemented is utilitarian. To use the VCG mechanism, we need to find the vertex cover with the minimum weight, which is NP-hard [28]. Since we are interested in mechanisms that can be computed in polynomial time, we must use polynomial-time computable output functions. Many algorithms have been proposed in the literature to approximate a minimum-weight vertex cover. In this chapter, we use a 2-approximation algorithm given in Ref. [28]. For completeness, we briefly review this algorithm here. The algorithm is round-based. Each round selects some vertices and discards some vertices. For each node i, $w(i)$ is initialized to its weight c_i, and when $w(i)$ drops to 0, i is included in the vertex cover. To make the presentation clear, we say edge (i_1, j_1) is *lexicographically smaller* than edge (i_2, j_2) if (1) $\min(i_1, j_1) < \min(i_2, j_2)$, or (2) $\min(i_1, j_1) = \min(i_2, j_2)$ and $\max(i_1, j_1) < \max(i_2, j_2)$.

Algorithm 83.2 outputs a vertex cover V' whose weight is within two times of the optimum. For convenience, we use VC(c) to denote the vertex cover computed by Algorithm 83.2 when the cost vector of vertices is c. Below we generalize Algorithm 83.2 to a more general scenario. Typically, a round-based output can be characterized as following Algorithm 83.3.

Definition 83.3

An updating rule$\mathcal{U}^r$ satisfies crossing-independence *if, for any agent i not selected in round r, (1) $\mathcal{S}^{r+1}$ and c_{-i}^{r+1} do not depend on c_j^r and (2) for fixed c_{-i}^r, $c_{i_1}^r \le c_{i_2}^r$ implies that $c_{i_1}^{r+1} \le c_{i_2}^{r+1}$.*

We have the following theorem on the existence of a truthful payment using the output function $\mathcal{A}$ defined by Algorithm 83.3.

Algorithm 83.2 Approximate Minimum-Weight Vertex Cover

```
Input: A node weighted graph $G = (V, E, c)$.
Output: A vertex cover $V'$.
Set $V' = \emptyset$. For each $i \in V$, set $w(i) = c_i$.
while $V'$ is not a vertex cover of $G$ do
    Pick an uncovered edge $(i, j)$ with the least lexicographic order among all uncovered edges.
    Let $m = \min(w(i), w(j))$.
    Update $w(i)$ to $w(i) - m$ and $w(j)$ to $w(j) - m$.
    If $w(i) = 0$, then add $i$ to $V'$.
    If $w(j) = 0$, then add $j$ to $V'$.
Output $V'$.
```

Algorithm 83.3 Compute a Round-Based Output Function $\mathcal{A}$

```
Input: A game $\mathcal{G}$, and the agents' cost vector $c$.
Output: An output $\mathcal{O}(c)$.
Set $r = 0$, $c^0 = c$, and $\mathcal{G}^0 = \mathcal{G}$ initially.
repeat
    Compute an output $o^r$ using a deterministic algorithm
        $$\mathcal{O}^r : \mathcal{S}^r \times c^r \rightarrow \{0, 1\}^n,$$
    where $\mathcal{O}^r$, $c^r$ and $\mathcal{S}^r$ are the output function, cost vector and game setting in game $\mathcal{G}^r$, respectively.
    Remark: For the example of vertex cover, $\mathcal{O}^r$ will always select the light-weighted node on the lexicographically least uncovered edge $(i, j)$.
    Let $r = r + 1$. Update the game $\mathcal{G}^{r-1}$ to obtain a new game $\mathcal{G}^r$ with setting $\mathcal{S}^r$ and cost vector $c^r$ according to a rule
        $$\mathcal{U}^r : \mathcal{O}^{r-1} \times (\mathcal{S}^{r-1}, c^{r-1}) \rightarrow (\mathcal{S}^r, c^r).$$
    Here we updates the cost and setting of the game.
until a valid output is found
Return the union of the set of the selected players of each round as the final output.
Remark. For the example of vertex cover, it is the union of nodes selected in all rounds.
```

Theorem 83.8

The output function $\mathcal{A}$ defined by Algorithm 83.3 satisfies MP if the output functions $\mathcal{O}^r$ satisfy MP and the updating function $\mathcal{U}^r$ satisfies crossing-independence for every round r.

Proof

Consider an agent i, and fix c_{-i}. We prove that if an agent i is selected with cost c_i, then it is also selected with cost $d_i < c_i$. Assume that i is selected in round r with cost c_i. Then under the cost vector $c|^i d_i$, if agent i is selected in a round before r, our claim holds. Otherwise, consider round r. The setting $\mathcal{S}^r$ and the costs of all other agents are the same as those when agent i had cost c_i since i is not selected in the previous rounds due to crossing-independence. Notice that if i is selected in round r with cost c_i, i is also selected in round r with $d_i < c_i$ because $\mathcal{O}^r$ satisfies MP. This finishes the proof. □

By Theorem 83.8, if the round-based output satisfies MP, the cut values exist. We next show how to find the cut value for a selected agent k in Algorithm 83.4.

Algorithm 83.4 Compute The Cut Values For A Round-Based Output Function $\mathcal{A}$

Input: A round-based output function $\mathcal{A}$, a game $\mathcal{G}^1 = \mathcal{G}$, an updating function vector $\mathcal{U}$, and the agents' cost vector c.
Output: The cut value x for agent k.

```
Set $r = 0$ and $c_k = \zeta$. Recall that $\zeta$ is a value such that $\mathcal{A}_k = 0$ when an agent reports the cost $\zeta$.
repeat
  Compute an output $o^r$ using a deterministic algorithm based on setting $\mathcal{S}^r$ using $\mathcal{O}^r : \mathcal{S}^r \times c^r \to \{0, 1\}^n$.
  Find the cut value for agent $k$ based on the output function $\mathcal{O}^r$ for costs $c^r_{-k}$. Let $\ell_r = \kappa_k(\mathcal{O}^r, c^r_{-k})$ be the cut value.
  Set $r = r + 1$, and obtain $\mathcal{G}^r$ from $\mathcal{G}^{r-1}$ and $o^r$ according to the updating rule $\mathcal{U}^r$.
  Let $c^r$ be the new cost vector for game $\mathcal{G}^r$.
until a valid output is found.
Let $g_i(x)$ be the cost of $c^i_k$ when the original cost vector is $c|^k x$.
Find the minimum value $x$ such that
    $\begin{cases} g_1(x) \geq \ell_1; \\ g_2(x) \geq \ell_2; \\ \vdots \\ g_{t-1}(x) \geq \ell_{t-1}; \\ g_t(x) \geq \ell_t. \end{cases}$
  Here, $t$ is the total number of rounds.
Output the value $x$ as the cut value.
```

To compute the cut values, we assume that (1) we can solve the equation $g_i(x) = b$ to find x in polynomial time when the cost vector c_{-i} and b are given; (2) the cut value ℓ_i for each round i can be computed in polynomial time.

Now we apply Algorithm 83.4 and Theorem 83.8 to the vertex cover problem. For each round r, we select a node with the least weight that is incident with the lexicographically least uncovered edge. The output function satisfies MP. For agent i, we update i's cost to $c^r_i - c^r_j$ if and only if edge (i, j) is selected. Observe that this updating rule satisfies crossing-independence. We can apply Algorithm 83.4 to compute the cut values as shown in Algorithm 83.5.

Algorithm 83.5 Compute Cut Value for MVC

Input: A node-weighted graph G and a node k selected by Algorithm 83.2.
Output: The cut value $\kappa_k(VC, c_{-k})$.

```
For each $i \in V$, set $w(i) = c_i$.
Set $w(k) = \infty$, $p_k = 0$, and $V' = \emptyset$.
while $V'$ is not a vertex cover do
  Pick an uncovered edge $(i, j)$ with the least lexicographic order among all uncovered edges.
  Set $m = \min(w(i), w(j))$.
  Update $w(i) = w(i) - m$ and $w(j) = w(j) - m$.
  If $w(i) = 0$, add $i$ to $V'$; else add $j$ to $V'$.
  If $i == k$ or $j == k$ then set $p_k = p_k + m$.
Output $p_k$ as the cut value $\kappa_k(VC, c_{-k})$.
```

83.5 General Demand Game

83.5.1 Characterize the Strategyproof Mechanism

For general demand games, we have a similar observation as the binary demand games. Here, we assume that the $O_i(c)$ is *piecewise continuous* with respect to any variable c_i for any agent i, that is, a finite number of piecewise linear functions. The only possible types of discontinuities for a piecewise continuous function are removable and step discontinuities. Recall the definition of MP does not restrict to the binary demand games: MP could be a property for general demand games as well. More interestingly, the MP is not only a necessary and sufficient condition for the existence of truthful mechanism in binary demand games, but also a necessary and sufficient condition for the existence of truthful mechanism in general demand games.

Theorem 83.9

For a given output function $\mathcal{O}$, there exists a payment scheme $\mathcal{P}$ such that the mechanism $M = (\mathcal{O}, \mathcal{P})$ is truthful if and only if $\mathcal{O}$ satisfies MP.

Proof

First, we prove that if there exists a strategyproof mechanism $M = (\mathcal{O}, \mathcal{P})$ then $\mathcal{O}$ satisfies MP. We consider two coefficients profile $\mathbf{d}|^i c_{i_1}$ and $\mathbf{d}|^i c_{i_2}$ where $c_{i_1} \le c_{i_2}$.

Consider the case when agent i has unit cost c_{i_1}. Recall that $\mathcal{P}$ is strategyproof, thus if agent i lies its unit cost to c_{i_2}, its utility should not increase. Thus, we have $\mathcal{P}_i(\mathbf{d}|^i c_{i_1}) - c_{i_1}\mathcal{O}_i(\mathbf{d}|^i a_{i_1}) \ge \mathcal{P}_i(\mathbf{d}|^i c_{i_2}) - c_{i_1}\mathcal{O}_i(\mathbf{d}|^i c_{i_2})$.

Now consider the case when agent i actually has cost coefficient c_{i_2}. Similarly, we have $\mathcal{P}_i(\mathbf{d}|^i a_{i_2}) - a_{i_2}\mathcal{O}_i(a|^i a_{i_2}) \ge \mathcal{P}_i(\mathcal{O}, a|^i a_{i_1}) - a_{i_2}\mathcal{O}_i(a|^i a_{i_1})$.

Combining the above two inequalities, we have $c_{i_2}[\mathcal{O}_i(\mathbf{d}|^i c_{i_1}) - \mathcal{O}_i(\mathbf{d}|^i c_{i_2})] \ge \mathcal{P}_i(\mathbf{d}|^i c_{i_1}) - \mathcal{P}_i(\mathbf{d}|^i c_{i_2}) \ge c_{i_1}[\mathcal{O}_i(\mathbf{d}|^i c_{i_1}) - \mathcal{O}_i(\mathbf{d}|^i c_{i_2})]$. Thus, we have $\mathcal{O}_i(\mathbf{d}|^i c_{i_1}) \ge \mathcal{O}_i(\mathbf{d}|^i c_{i_2})$ as $c_{i_1} \le c_{i_2}$. This proves that $\mathcal{O}$ satisfies MP.

To prove that if $\mathcal{O}$ satisfies MP then there exists a strategyproof payment $\mathcal{P}$ by construction. For an agent i, we first fix $\mathbf{d}_{-i}$ and use x to denote cost vector $\mathbf{d}|^i x$ if no confusion is caused. From the assumption that $\mathcal{O}$ satisfies MP, function $\mathcal{O}_i(x)$ is nonincreasing. Recall that $\mathcal{O}_i(x)$ is a piecewise continuous function. We let $x_1 < x_2 < \cdots < x_m$ be the points at which $\mathcal{O}_i(x)$ is not continuous, and introduce a dummy point $x_{m+1} = \infty$. We define a function $\kappa_i(x)$ such that, for $x_p < x \le x_{p+1}$,

$$\kappa_i(x) = x\mathcal{O}_i(x) + \int_x^{x_{p+1}} \mathcal{O}_i(y)\,dy + \sum_{j=p+1}^{m} \int_{x_j}^{x_{j+1}} \mathcal{O}_i(y)\,dy$$

Given an output function $\mathcal{O}$ and a declared cost vector $\mathbf{d}$, Algorithm 83.6 defines the payment based on algorithm $\mathcal{O}$.

Thus, we only need to prove the payment scheme computed by Algorithm 83.6 is truthful. The proof is omitted here, refer to paper [29] for more details. □

Algorithm 83.6 Payment Scheme based on $\mathcal{O}$

Input: Algorithm $\mathcal{O}$ and vector $\mathbf{d}$.
Output: The payment scheme $\mathcal{P}$.

```
for each agent $i$ do
    Fix $\mathbf{d}_{-i}$. The payment to $i$ is $\mathcal{P}_i(\mathbf{d}) = \kappa_i(d_i)$.
```

If we specify that if an agent i has payment 0 when $O_i(\mathbf{d}) = 0$ (which is called *normalized* payment scheme), then we have the following theorem:

Theorem 83.10

Given an algorithm $\mathcal{O}$ satisfying MP, the payment scheme defined by Algorithm 83.6 is the only *normalized truthful payment scheme.*

With Theorem 83.9 and Algorithm 83.6, one could design the truthful payment scheme if it exists. However, the truthful payment scheme may not exist at the first place. Thus, one need to modify the existing approximation algorithm to make it monotonic while still keep the approximation ratio if possible. Following we use DiffServ Multicast Game as an example to show how we can achieve this under certain circumstance.

83.5.2 DiffServ Multicast Game

In this section, we use the DiffServ multicast game as an example to show how to design the truthful mechanisms. We first show that the previous approximation algorithm does not satisfy the monotonic property and as a result there does not exist truthful mechanism. In light of this negative result, we modify the approximation algorithm such that it satisfies monotonic property and still achieves the same approximation ratio.

Network model and problem statement. We assume that there is a connected network $G = (V, E)$ with vertex set V, edge set E, where $|V| = n$ and $|E| = m$. Every edge e_i has a cost function $c_i x$ if x is the bandwidth e_i dedicated to a multicast transmission. There is a source node s and a set of receivers $R \subset V$ that request to receive the multicast service. Every receiver $r_i \in R$ has a bandwidth demand h_i that specifies the minimum bandwidth it needs. The DiffServ multicast problem consists of two parts: (1) finding a tree rooted at the sender s that spans all receivers in the receiver set and (2) find a bandwidth reservation for each link for this multicast. The tree topology and bandwidth reservation should satisfy that for any receiver r_i, each link on the tree path between r_i and s has a bandwidth reservation not smaller than d_i. Thus, for a link e_i, the reserved bandwidth should not be smaller than the maximum bandwidth demand of its downstream receivers. The weight of a multicast topology T with link bandwidth reservation vector $b = \{b_1, b_2, \ldots, b_m\}$ is $\omega(T, b) = \sum_{e_i \in T} c_i b_i$. Given the cost vector $\mathbf{c}$ of all links and the bandwidth demand $\mathbf{h}$ of all receivers, the DiffServ multicast problem is to construct a tree T and a bandwidth reservation b with the minimum cost $\omega(T, b)$.

Approximation algorithm. The high level idea of the algorithm to construct the DiffServ Multicast Tree is as follows. The receiver set is divided into subsets, each containing receivers with demands in a particular range. These subsets are handled in multiple rounds, in a descending order according to their bandwidth demand ranges. In each round, the demands of all receivers in a subset are treated equally and we apply the 2-approximation algorithm by Ref. [30] for a general link-weighted Steiner tree. The new Steiner tree is connected to the DiffServ multicast tree being built and the links picked in earlier rounds are set to cost 0 for later rounds. Following Algorithm 83.7 illustrates the details.

Theorem 83.11

Algorithm 83.7 constructs a tree whose weight is at most eight times the weight of the minimal cost DiffServ multicast tree T^{opt}.

However, as shown in the following lemma, Algorithm 83.7 does not satisfy the MP property, which implies that there does not exist truthful mechanism if we use Algorithm 83.7 as the output of the mechanism.

Lemma 83.3

Algorithm 83.7 does not satisfy MP.

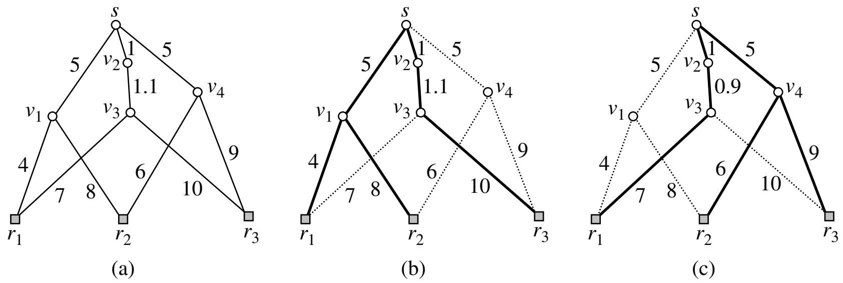


FIGURE 83.2 The spanning tree constructed by Algorithm 83.7. (a) original G, (b) tree when $e_2 = 1.1$, and (c) tree when $e_2 = 0.9$.

Algorithm 83.7 Construct DiffServ Multicast Tree

Input: A network G with cost vector $\mathbf{c}$, a source node s, a set of receivers R and a bandwidth demand vector $\mathbf{h}$.

Output: A tree DMT and a bandwidth allocation vector B.

```
Sort all receivers according to their bandwidth demands in an descending order, say $R = \{r_1, r_2, \ldots, r_k\}$.
Initialize the tree $T$ to empty and index $t = 1$.
for each link $e_i$ do
    Label it as WHITE and set $B_i = 0$.
repeat
    Let $r_j$ be the first receiver in the receiver set $R$.
    Find the maximal index $k$ such that $h_k \geq \frac{h_j}{2}$.
    Set $d_i = 0$ for each BLACK link and $d_i = b_i \cdot c_i$ for each WHITE link.
    Let $R_t = \{r_j, \ldots, r_k\}$ and find the spanning tree $T_t = LST(R_t, \mathbf{d})$.
    Remove $R_t$ from $R$ and mark all links in tree $T_t$ as BLACK.
    Set $T = T \bigcup T_t$ and $t = t + 1$.
until the receiver set $R$ is empty.
for each link $e_i \in T$ do
    Find $e_i$'s downstream receiver with the maximum bandwidth demand, say $q_j$ and set $B_i = h_j$.
Output $T$ as $DMT$ and bandwidth vector $B$.
```

Proof

We prove it by presenting an example here. A network G has three receivers r_1, r_2, r_3 with bandwidth demand $d_1 = d_2 = 1$ and $d_3 = 2$. The unit costs of the links are shown in Figure 83.2(a). When we apply Algorithm 83.7 to network G, we obtain a tree shown in Figure 83.2(b). Let agent 2 be link v_2v_3. The bandwidth allocation of link $e_2 = v_2v_3$ is 2. Consider the scenario when the unit cost of link e_2 changes from 1.1 to 0.9 while other unit costs remain the same. The new spanning tree topology constructed by Algorithm 83.7 is shown in Figure 83.2(c). The bandwidth allocation of e_2 becomes 1, which decreases by half compared with the bandwidth reservation with coefficient 1.1. This finishes our proof. □

New monotonic algorithm. In light of the negative result of Algorithm 83.7, we modify Algorithm 83.7 as follows to obtain Algorithm 83.8 that satisfies the MP property which still has the approximation ratio 8.

Theorem 83.12

Algorithm 83.8 constructs a tree whose weight is at most eight times the weight of the minimal cost DiffServ multicast tree T^{opt} and satisfies MP.

Algorithm 83.8 Construct New DiffServ Multicast Tree

Input: A network G with coefficient vector a, a source node s, a set of receivers R and a bandwidth demand vector d.

Output: A spanning tree $\overline{DMT}$ and a bandwidth allocation vector $\overline{B}$.

```
Sort all receivers according to their bandwidth demands in an descending order, say $R = \{r_1, r_2, \ldots, r_k\}$.
Initialize the tree $T$ to empty and index $t = 1$.
for each link $e_i$ do
  Label it as WHITE and set $\overline{B}_i = 0$.
repeat
  Let $r_j$ be the first receiver in the receiver set $R$.
  Find the maximal index $k$ such that $d_k \geq \frac{d_j}{2}$.
  Set the cost of each WHITE link $e_i$ as $c_i = a_i \cdot d_j$ and each BLACK link as 0.
  Let $R_t = \{r_j, \ldots, r_k\}$ and find the spanning tree $T_t = LST(R_t, c)$ using Algorithm in [30].
  Remove $R_t$ from $R$ and mark all links in tree $T_t$ as BLACK.
  Set $T = T \bigcup T_t$.
  for each link $e_i \in T_t$ do
    If $\overline{B}_i = 0$ then set $\overline{B}_i = d_j$.
  Set $t = t + 1$.
until the receiver set $R$ is empty.
Output $T$ as $\overline{DMT}$ and bandwidth vector $\overline{B}$.
```

With Theorem 83.12, we can apply the general framework 83.6 to obtain the truthful mechanism. The details of the truthful mechanism depends on the specific structure of the tree $\overline{DMT}$ and is omitted here. Interesting reader can refer to paper [29] for more details.

83.6 Literature Review

In this section, we review the literatures in the designing of truthful mechanism for the approximation algorithms.

Designing strategyproof mechanisms, when an approximated algorithm to the objective function is used, was studied for some *specific* problems [1,8,31,32]. Nisan and Ronen [14] studied the strategyproof mechanism for the job scheduling. Auletta et al. [33] studied deterministic strategyproof approximation schemes for scheduling on related machines. Lehmann et al. [24] studied the combinatorial auctions. Archer et al. [34] studied the approximate strategyproof mechanism for combinatorial auctions with single-parameter agents. Devanur et al. [35] studied the strategyproof cost-sharing mechanisms for set cover and facility location games.

Lehmann et al. [24] studied how to design an efficient truthful mechanism for single-minded combinatorial auction. In a single-minded combinatorial auction, each agent i $(1 \leq i \leq n)$ only wants to buy a subset $S_i \subseteq S$ with private price c_i. A single-minded bidder i declares a bid $b_i = \langle S_i', a_i \rangle$ with $S_i' \subseteq S$ and $a_i \in R^+$. In Ref. [24], it is assumed that the set of goods allocated to an agent i is either S_i' or $\emptyset$, which is known as *exactness*. Lehmann et al. gave a greedy round-based allocation algorithm that has an approximation ratio $\sqrt{m}$, where m is the number of goods in S. On the basis of the approximation algorithm, Lehmann et al. gave a truthful payment scheme. For an allocation rule satisfying (1) exactness: the set of goods allocated to an agent i is either S_i' or $\emptyset$; (2) monotonicity: bidding more money for fewer goods cannot cause a bidder to lose its bid, they proposed a truthful payment scheme as follows: (1) pay a winning bidder the amount that does not depend on its own bidding; and (2) pay a losing bidder 0.

Mu'alem and Nisan [23] further generalize this idea and proposed several combination algorithms including MAX, IF–THEN–ELSE construction to perform partial search. All of their algorithms required the welfare function associated with the output to satisfy a certain *bitonic* property. Kao et al. [19] further generalize this direction for any binary demand game in which an agent is either selected or not selected. An output method is said to be *monotonic* if, given fixed valuations of all other agents, there exists a threshold value κ_i for an agent i such that the agent i is selected if and only if its valuation is at least κ_i. Interesting, as shown by Kao et al. [19], the MP is still a necessary and sufficient condition for the existence of truthful mechanism in this broader setting. Furthermore, an approximation algorithm whose output is monotone and the payment scheme pays a selected agent its threshold value is truthful.

The demand game is a natural generalization of binary demand game and has also been studied in literatures. Archer and Tardos [36] showed how to design truthful mechanisms for several combinatorial problems where each agent's private information is naturally expressed by *a single positive real number*, which will always be the cost incurred per unit load. The mechanism's output could be an arbitrary real number but their valuation is a quasilinear function tw, where t is the private per unit cost and w the work load. Archer and Tardos characterized that all truthful mechanisms should have decreasing "work curves" w and that the truthful payment should be

$$P_i(b_i) = P_i(0) + b_i w_i(b_i) - \int_0^{b_i} w_i(u)\,\mathrm{d}u$$

Using this model, Archer and Tardos designed the truthful mechanisms for several scheduling related problems, including minimizing the span, maximizing flow, and minimizing the weighted sum of completion time problems.

Recently, people are studying how to transfer an approximation algorithm that is not monotonic into an approximation algorithm that is monotonic without loss of much approximation ratio. Briest et al. [37] first showed that the most basic techniques for the approximation algorithm does not satisfy the monotonic property: the transformation of a pseudopolynomial-time algorithm into a fully polynomial-time approximation algorithm scheme (FPTAS); the distinction of elements according to size of the parameters as often used in the design of the polynomial-time approximation scheme (PTAS); and randomized rounding for packing integer programs (PIPS). In light of these observations, they provide some new techniques that result in some monotonic approximation algorithm for FPTAS, PTAS, and PIPS.

83.7 Conclusion

In this chapter, we mainly discuss how to design the truthful mechanism for the approximation algorithms. First of all, we show that if we use the approximation algorithms instead of the optimal solution, then it is almost universal that the celebrated VCG mechanism is not truthful anymore. In light of this failure, some mechanisms other than VCG mechanisms are needed to address these issues. We study how to design truthful mechanisms for binary demand games in Section 83.4 and more generalized demand game in Section 83.5. The monotonic property for approximation algorithm is a necessary and sufficient condition for the existence of truthful algorithm and present some general techniques to compute the payment scheme for the monotonic approximation algorithms. However, one important question that has not been explored fully is that how we can convert an approximation algorithm that does not satisfy the monotonic property into an approximation algorithm that satisfies the monotonic property with same or similar approximation ratio. Briest et al. [37] made the progress in this direction by presenting some general techniques to design some FPTAS, PTAS, and PIPS algorithms.

We end this chapter by pointing out some possible future directions in the mechanism design for the approximation algorithms. First of all, there are some algorithms that are beyond the demand games, that is, there may have more than one parameter to be optimized. It is interesting to study how to design truthful mechanism for the bicriteria approximation algorithms. Second, it is of great importance if we analyze the algorithms that do not satisfy the monotonic property and design general techniques to convert them into the approximation algorithms that satisfy the monotonic property.

Acknowledgment

Partially supported by NSF CCR-0311174.

References

[1] Roughgarden, T. and Tardos, E., How bad is selfish routing? *Proc. of FOCS,* 2000, p. 93.

[2] Shenker, S., Making greed work in networks: a game-theoretic analysis of gateway service disciplines, *Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems,* 1990, p. 241.

[3] Cocchi, R., Shenker, S., Estrin, D., and Zhang, L., Pricing in computer networks: motivation, formulation, and example, *IEEE/ACM Trans. Networking,* 1, 614, 1993.

[4] Orda, A., Rom, R., and Shimkin, N., Competitive routing in multiuser communication networks, *IEEE/ACM Trans. Networking,* 1(5), 510, 1993.

[5] Shenker, S., Clark, D., Estrin, D., and Herzog, S., Pricing in computer networks: reshaping the research agenda, *ACM Comput. Commun. Rev.,* 26, 19, 1996.

[6] Park, K., Sitharam, M., and Chen, S., Quality of service provision in noncooperative networks: heterogenous preferences, multi-dimensional qos vectors, and burstiness, *Proc. Int. Conf. on Information and Computation Economies,* 1998, p. 111.

[7] Wang, X. and Schulzrinne, H., Pricing network resources for adaptive applications in a differentiated services network, *Proc. of INFOCOM*, 2001.

[8] Marbach, P., Pricing differentiated services networks: bursty traffic, *Proc. of INFOCOM*, 2001.

[9] Cao, X.-R., Shen, H.-X., Milito, R., and Wirth, P., Internet pricing with a game theoretical approach: concepts and examples, *IEEE/ACM Trans. on Networking,* 10(2), 208, 2002.

[10] Lomonosov, A., Sitharam, M., and Park, K., Stability vs optimality tradeoff in game theoretic mechanisms for qos provision, *Proc. ACM Symp. on Applied Computers*, 2003, p. 28.

[11] Dutta, D., Goel, A., Govindan, R., and Zhang, H., The design of a distributed rating scheme for peer-to-peer systems, *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[12] Dellarocas, C. and Resnick, P., Online reputation mechanisms: a roadmap for future research, *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[13] Marbach, P., Analysis of a static pricing scheme for priority services, *IEEE/ACM Trans. on Networking,* 12(2), 312, 2004.

[14] Nisan, N. and Ronen, A., Algorithmic mechanism design, *Proc. of STOC,* 1999, p. 129.

[15] Clarke, E. H., Multipart pricing of public goods, *Public Choice,* 11, 17, 1971.

[16] Groves, T., Incentives in teams, *Econometrica*, 41, 617, 1973.

[17] Vickrey, W., Counterspeculation, auctions and competitive sealed tenders, *J. Finance,* 16, 8, 1961.

[18] Nisan, N. and Ronen, A., Computationally feasible VCG mechanisms, *ACM Conf. on Electronic Commerce,* 2000, p. 242.

[19] Kao, M.-Y., Li, X.-Y., and Wang, W., Towards truthful mechanisms for binary demand games: a general framework, *ACM Conf. on Electronic Commerce*, 2005.

[20] Wang, W., Li, X.-Y., Sun, Z., and Wang, Y., Design multicast protocols for non-cooperative networks, *Proc. of INFOCOM,* 2005.

[21] Green, J. and Laffont, J. J., Characterization of satisfactory mechanisms for the revelation of preferences for public goods, *Econometrica*, 45, 427, 1977.

[22] Chvatal, V., A greedy heuristic for the set covering problem, *Math. Oper. Res.,* 4(3), 233, 1979.

[23] Mu'alem, A. and Nisan, N., Truthful approximation mechanisms for restricted combinatorial auctions: extended abstract, *Proc. Nat. Conf. on Artificial Intelligence*, 2002, p. 379.

[24] Lehmann, D., O'Callaghan, L. I., and Shoham, Y., Truth revelation in approximately efficient combinatorial auctions, *JACM,* 49(5), 577, 2002.

[25] Lehmann, D. J., O'Callaghan, L. I., and Shoham, Y., Truth revelation in approximately efficient combinatorial auctions, *ACM Conf. on Electronic Commerce*, 1999, p. 96.

[26] Hochbaum, D. S., Efficient bounds for the stable set, vertex cover, and set packing problems, *Discrete Appl. Math.,* 6, 243, 1983.

[27] Robins, G. and Zelikovsky, A., Improved Steiner tree approximation in graphs, *Proc. SODA,* 2000, p. 770.

[28] Bar-Yehuda, R. and Even, S., A local-ratio theorem for approximating the weighted vertex cover problem, *Ann. Discrete Math.,* 25, 26, 1985.

[29] Wang, W., Li, X.-Y., and Sun, Z., Design differentiated service multicast with selfish agents, *Proc. Int. Conf. on Algorithmic Applications in Management,* 2005.

[30] Takahashi, H. and Matsuyama, A., An approximate solution for the Steiner problem in graphs, *Math. Japonica,* 24, 573, 1980.

[31] Fiat, A., Goldberg, A., Hartline, J., and Karlin, A., Competitive generalized auctions, *Proc. of STOC*, 2002.

[32] Goemans, M. and Sketella, M., Cooperative facility location games, *Proc. of SODA*, 2000.

[33] Auletta, V., Prisco, R. D., Penna, P., and Persiano, P., Deterministic truthful approximation schemes for scheduling related machines, *Proc. Int. Workshop on Interconnection Networks,* 2003.

[34] Archer, A., Papadimitriou, C., Talwar, K., and Tardos, E., An approximate truthful mechanism for combinatorial auctions with single parameter agents, *Internet Math.,* 1(2), 129, 2003.

[35] Devanur, N. R., Mihail, M., and Vazirani, V. V., Strategyproof cost-sharing mechanisms for set cover and facility location games, *Proc. of ACM EC*, 2003.

[36] Archer, A. and Tardos, E., Truthful mechanisms for one-parameter agents, *Proc. of FOCS,* 2001, p. 482.

[37] Briest, P., Krysta, P., and Vocking, B., Approximation techniques for utilitarian mechanism design, *Proc. of STOC,* 2005, p. 39.

84

Histograms, Wavelets, Streams, and Approximation

Sudipto Guha
University of Pennsylvania

84.1 Introduction

Over the past decade, the size of data seen by a computational problem has grown immensely. There appears to be more web pages than human beings, and we have successfully indexed the pages. Routers generate huge traffic logs, in the order of terabytes, in a short time. The same explosion of data is felt in observational sciences because our capabilities of measurement have grown significantly. In comparison, computational resources have not increased at the same rate. In particular, it has been found that the ability of random access to data itself is a resource. In some settings, each individual input item is not so significant by itself—consider monitoring a network, estimating costs of query plans, etc.—but the quantity we are interested in is the aggregate picture that emerges from the data. In several scenarios, such as network monitoring, some data are never stored but merely used to infer aggregate health of the network. At the same time, in several data-intensive computations, making passes over the data has been found to be significantly more efficient. This has brought the data stream model to the fore. The model consists of an algorithm with a small random access memory, typically sublinear in input size, operating in passes over the input. Any input item not explicitly stored is inaccessible to the algorithm in the same pass. The model captures the essence of a monitoring process that is allowed to observe a system unobtrusively using some small "extra" space. Of particular interest is the one pass model, where the input may simply not be stored at all. The data stream model poses several challenges. Intriguingly, even simple problems, which were thought to be fully understood at small scales, have been found to be ominous at the current scale of data and have required reexamination. As mentioned earlier, the aggregate picture that emerges from the the data, or the synopsis, is often the desiderata.

The idea of synopses is not new. We can view the data as a function over a suitable domain. Expressing a function accurately as a combination of a few simpler functions has been at the heart of approximation theory in mathematics, and dates back centuries starting with polynomial approximations and the work of Fourier. Histograms and bar charts have been in use since the middle ages. The Haar system was proposed as early as 1910. Initially, the thrust of synopsis construction had been to project the data on to a fixed space. The benefit of such schemes is that the sum of two synopses is the synopsis of the sum of the original functions. This had led to the bulk of work in mathematical approximation theory, known as linear approximation theories. The theorems proved about these were largely extremal, namely, projections that work for all data: what is the maximum error (again considering all data) given a fixed projection strategy, etc. Schmidt, in 1909, was one of the earliest to consider nonlinear theories where the image space is dependent on data; which in modern terms can be viewed as a data-dependent or data-driven synopsis. This immediately raised the question of approximating the given data in the best possible way. This question is closer to common optimization problems—and since the end goal is approximation, it is only natural to consider approximation algorithms for these problems. Most synopses techniques used currently are in the nonlinear category.

In the context of databases, synopses date back to Selinger et al. [1] in the late 1970s. They proposed estimating the cost of various operators using synopses of data to decide between alternative query plans. The first synopsis structure proposed for this purpose was a simple division of the domain into equal sizes. Over time, it was recognized that piecewise constant approximations of the data, or serial histograms, are significantly more accurate descriptions of distributions. Subsequently, it was demonstrated that the ℓ_2 distance between the representation and the data was an accurate estimate, which brought histograms closer to the mathematical definition of approximating functions. Since the late 1980s, wavelets have became popular as a tool in image processing. Their success was primarily due to the existence of fast algorithms for transforms and their multiresolution nature. They were introduced in databases in the context of "data cubes" to describe the data hierarchically. Wavelets and histograms are, by no means, the only synopses structures used. Quantiles and other estimates have been used as well. We will only consider histograms and wavelet approximations of data in this chapter.

Our goal in this chapter will not be to catalog the problems and the best results known. Our main aim is to introduce the reader to these problems and demonstrate what style of analysis is used. To that end we will consider the simpler versions of the problems, and restrict ourselves to one dimension mostly. The problems we will focus on will be illustrative and will not be exhaustive. The notes at the end of the chapter contain pointers to the more commonly known variants of these problems and the respective references. We will only refer to works on histograms and wavelets in the main body of the chapter. The sources of the algorithms discussed can be found in the notes. We will focus on those problems that are the simplest to state and are yet nontrivial to solve in massive data set context, and therefore are the basic problems in this area. We subsequently discuss histograms and then wavelets. We conclude with notes on the literature in this area.

84.2 Definitions and Problems

Definition 84.1 (Data Streams)

A data stream is a model of computation that treats random access as a resource.

In particular, given a set of m objects $Y = Y(1), \ldots, Y(m)$ we want to compute a function $f(Y)$ using small space and one pass over the data under the following restrictions: (i) the items $Y(j)$ are inspected in an increasing order of j, and (ii) any item not explicitly stored is "forgotten"—we do not know its value any longer (in the same pass). the computation proceeds in passes over the data. Unless otherwise mentioned, a data stream algorithm will refer to a one pass streaming algorithm. The streaming models differ in the *semantics* of the stream items $Y(j)$. There is a multitude of models, but we will discuss the two most common ones: (1) We can have $Y(j) = X(j)$ and $m = n$, where we are considering a function $X(i)$ defined on integers, which is specified in an increasing order of (i). This is a **Time series model**. (2) We can have $Y(j) = \langle i, \delta_j \rangle$ where $i \in [0, n]$ for some n. Each element $Y(j)$ implies "set $X(i) = X(i) + \delta_j$" (δ_j can be negative) and thus specifies a function X as a sequence of updates. This is the **Update model**; also known as the cash-register/dynamic model.

In this chapter, we mainly focus on the time series model of data streams. This model is the simplest stream model and relevant in the context of sensor data, stocks, etc., where the order in which the data arrives has a natural meaning. A good sublinear space algorithm for this model implies a good algorithm with small "extra" space—as is typical in the monitoring setting. Concrete examples of such systems are the "self-tuning" systems ([2]) where the system executes queries and a monitoring component gathers information about various parameters to optimize/maintain the system performance. In essence, the input to the monitoring process is "free" because that input to the monitoring system is the result of some computation that was necessary anyways. An example in this context is the work of Bruno et al. [3], who consider learning histograms from observing answers to database queries. Any resource allocated to the monitor implies less resource for the actual system and it is naturally desired that the monitor has a small footprint.

Although the above is true for the update model of streams, the maintenance of relevant information under updates is often a bigger challenge than solving the original problem from the maintained information. These algorithms typically find nontrivial ways of capturing similar computation as in a time-series model stream. The techniques used in these algorithms are exciting, but orthogonal to the question of histogram construction.

In summary, the time series model captures the problem at an abstract level that is restricted compared with offline computation, but rich enough to allow us to design interesting algorithms. Very often, these algorithms are the stepping stones to the most general results on update streams. In the interest of space, we omit discussing the results in the update stream model, but the notes contain references to them. In this chapter, we focus on the following problems:

Problem 84.1 (HISTOGRAMS)

Given a set of numbers $X = X(0), \ldots, X(n-1)$ in a (time series) stream, find a piecewise constant function H with at most B pieces to minimize some suitable function of the error $X - H$, e.g., $\|X - H\|_2$, $\|X - H\|_\infty$, etc. Each "piece" corresponds to a subinterval $[a, b]$ of $[0, n-1]$ and is represented by one number. Unless otherwise specified, we would assume that the B pieces induce a partition of the interval $[0, n-1]$.

Each of the pieces is defined as a "bucket." Given an i, we find the bucket to which i belongs to and return the representative number, say v, for the bucket as an estimate of $X(i)$. The error introduced by this process is $X(i) - v$, which can be viewed as $X(i) - H(i)$. A natural goal of any accurate description would be to minimize a suitable function of $X - H$. One of the most natural measures is the ℓ_2 norm (or its square) of the error vector $X - H$; however, ℓ_1, ℓ_∞ measures are common as well. We can also consider weighted variants where given a weight vector $\{\pi_i\}$ we seek to minimize a suitable function of the terms $\pi_i(X(i) - H(i))$. The weighted variants are sometimes termed "workload optimal." Several questions arise immediately—are the intervals allowed to overlap, should they cover the entire $[0, n-1]$, etc. In the case of overlapping intervals, it is unclear how to define the value of a point that belongs to two buckets. However, under any definition, we can easily see that B overlapping buckets define at most $2B - 1$ nonoverlapping buckets over any interval. A natural extension of the above is to piecewise polynomials. These have a rich history in numerical estimation algorithms. A priori, it is not clear why we should be able to achieve near optimal solutions for these problems in near linear time, which brings us back to the motivation of studying time series models.

Problem 84.2 (PIECEWISE POLYNOMIALS)

Solve the above problem of expressing a function using B nonoverlapping pieces where the pieces are small degree polynomials.

We can pose a problem about wavelets analogously, and about the connections:

Problem 84.3 (WAVELETS)

Given a wavelet basis $\{\psi_i\}$ and a set of numbers $X = X(0), \ldots, X(n-1)$ in a data stream, find a set of values $Z(i)$ with at most B nonzero values such that a suitable error of $X - \sum_i Z(i)\psi_i$ is minimized.

Problem 84.4 (Connections between Synopses)

What are the connections between the various synopses and can we leverage them to devise better algorithms?

The above problems do not make an explicit assumption on the dimension of the data set. For wavelets, there are very few changes in the results in the offline setting. Histograms, in contrast, turn out to be NP-hard, primarily due to two-dimensional partitioning. The issue of overlap of buckets becomes critical, since the number of nonoverlapping buckets required to express B overlapping buckets is exponential in the dimension.

Further, two- or more dimensional streaming is tricky to define except in update streams. The time series model is implicitly one-dimensional, in the special dimension that corresponds to the semantics of time. In this chapter, we will focus on the one-dimensional case and point the reader to the specific papers for the higher dimensional case.

The above, by no means, is an exhaustive list of interesting problems. However, the above are indeed *basic* in the sense that they are simple to pose and not always easy to solve in sublinear space.

84.3 Histograms

As mentioned earlier, histograms are piecewise constant approximations of data. Recall that the histogram problem is defined as follows: Given a set of numbers $X = X(0), \ldots, X(n-1)$ in a streaming fashion, find a piecewise constant function, H with at most B pieces to minimize some suitable function of the error $X - H$, e.g., $||X - H||_2$, $||X - H||_\infty$, etc. We will assume that $X(i)$ are polynomially bounded integers, since the histograms are most often used to approximate frequency. The discussion will extend to reals provided the minimum nonzero error of estimation using a histogram can be bounded from below, which is also a finite precision assumption. We will focus on the ℓ_2^2 measure. Using this as an example, we will see how to construct faster approximation algorithms. Subsequently, we will see how to extend the result to measures similar to ℓ_∞. All the discussion extends to weighted variants using standard techniques.

84.3.1 The Vopt or the ℓ_2^2 Measure

The measure is popular in databases and is also interesting mathematically. In this problem, the interval $[0, n-1]$ is partitioned into B pieces.

Observation 84.1

Due to the partitioning, we can express the ℓ_2^2 error as a sum of bucket errors. In each bucket, the best representative is the mean/average of the numbers.

Let Terr$[i, k]$ be the minimum ℓ_2^2 error of approximating $[0, i]$ using at most k buckets. Terr$[i, k]$ is computed for the points in $[0, i]$ only. A natural dynamic program (DP) that tries all possible guesses of the last interval $[j+1, i]$ is immediate. If the ℓ_2^2 error of approximating the interval $[j+1, i]$ by its mean is Sqerror$(j+1, i) = \sum_{r=j+1}^{i} X(r)^2 - (\sum_{r=j+1}^{i} X(r))^2/(i-j)$, we have Terr$[i, k] = \min_j \{$Terr$[j, k-1] +$ Sqerror$(j+1, i)\}$.

The final solution is given by Terr$[n, B]$. Maintaining the prefix sums $\sum_{r=0}^{j} X(r)$, $\sum_{r=0}^{j} X(r)^2$, the values of Sqerror$(j+1, i)$ can be computed in $O(1)$ time. Immediately, we arrive at an $O(n^2 B)$ algorithm using $O(nB)$ space. The space requirement can be reduced to $O(n)$, but a natural question arises: "Since the primary role of histograms is in approximating data, can we develop linear time algorithms that are near optimal approximations of the best histogram?" In what follows, we show how to achieve such an algorithm. The starting point is as follows.

Observation 84.2

*T*Err$[j, \cdot]$ *is nondecreasing and* Sqerror$(j, \cdot)$ *(and therefore* Sqerror$(j+1, \cdot)$*) is non-increasing.*

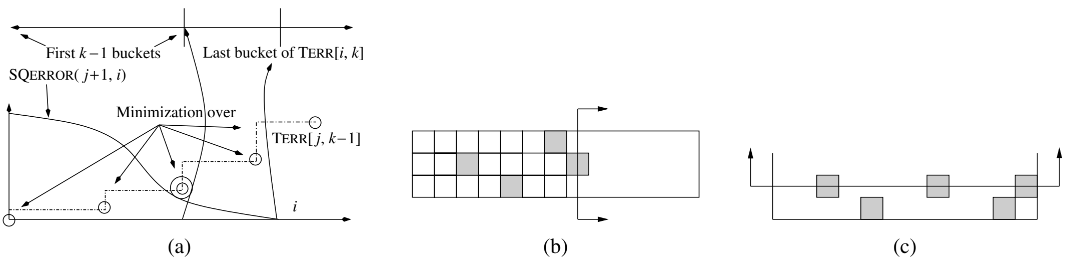


FIGURE 84.1 (a) Approximating $\textsc{Terr}[i, k-1]$ by a histogram, (b) front moving left to right, and (c) front moving bottom up.

It may appear that we can immediately use the above properties to get faster algorithms, but that is not the case. Consider $v_1, \ldots, v_n$ where each $v_i \geq 0$. Let $f(i) = \sum_{r=1}^{i} v_r$, and $g(i) = f(n) - f(i-1)$. The function $f(i)$ is nondecreasing and $g(i)$ is nonincreasing. But finding the minimum of $f(i) + g(i)$ amounts to minimizing $f(n) + v_i$, or in other words minimizing v_i. Note that this does not rule out that over B levels, the cost of the searching can be amortized—but no such analysis exists to date. The interesting aspect of the example is that picking any i gives us a 2 approximation (since $f(n) + v_i \leq 2f(n)$ and the minimum is no smaller than $f(n)$). In essence, the searching can be reduced if we are willing to settle for an approximation.

The central idea is that instead of storing the entire function $\textsc{Terr}[j, k-1]$, we approximate the function as shown in Figure 84.1. The interval $[1, i]$ is broken down into τ intervals (a_u, b_u) to approximately represent the function with a "staircase." We have $a_1 = 1$, $a_{u+1} = b_u + 1$, and $b_\tau = n$. Furthermore, the intervals are created such that the value of the function at the right hand boundary of an interval is at most a factor $(1 + \delta)$ times the value of the function at the left hand boundary. The number of maximum such intervals is $O(\frac{1}{\delta} \log n)$.

Now for any $a_u \leq j \leq b_u$, we have $(1+\delta)\textsc{Terr}[a_u, k-1] + \textsc{SqError}(a_u+1, i) \leq (1+\delta)\textsc{Terr}[j, k-1] + \textsc{SqError}(j+1, i)$ and which in turn is at most $\textsc{Terr}[b_u, k-1] + \textsc{SqError}(b_u+1, i)$; this rewrites to $(1+\delta)\textsc{Terr}[i, k] \leq \textsc{Terr}[b_u, k-1] + \textsc{SqError}(b_u+1, i)$. This implies that evaluating the sum at b_u gives us a $(1+\delta)$ approximation. However, there is a caveat—we cannot simultaneously approximate $\textsc{Terr}[i, k]$ and assume that we know $\textsc{Terr}[j, k-1]$ exactly for all $j < i$, $k > 2$. The solution is to employ the "ostrich algorithm," i.e., ignore the issue and simply use the approximation $\textsc{ApxErr}[j, k-1]$ (of $\textsc{Terr}[j, k-1]$) to compute $\textsc{ApxErr}[i, k]$. We can show by induction that the ratio of $\textsc{ApxErr}[i, k]$ to $\textsc{Terr}[i, k]$ is at most $(1+\delta)^{k-1}$. Setting $\delta = \frac{\epsilon}{2B}$ gives us a $(1+\epsilon)$ approximation since $(1 + \frac{\epsilon}{2B})^B \leq 1 + \epsilon$ for $\epsilon \leq 1$. The benefit of the algorithm is that we evaluate the sum at $O(\frac{1}{\delta} \log n)$ points assuming that the input integers are polynomially bounded. The entire algorithm runs in $O(\frac{nB}{\delta} \log n)$ time. The algorithm proceeds from left to right, and as more data arrive the function $\textsc{Terr}[i, k]$ does not change and the staircase we have constructed remains valid. This, along with the fact that we only need to store $\sum_{r=1}^{b_u} X(r)$, $\sum_{r=1}^{b_u} X(r)^2$ for the points b_u allow the algorithm to be an $O(\frac{B^2}{\epsilon} \log n)$ space streaming algorithm. Therefore:

Theorem 84.1

We can compute a $(1+\epsilon)$ approximation of the optimal histogram under ℓ_2^2 error over a data stream in $O(\frac{nB^2}{\epsilon} \log n)$ time and $O(\frac{B^2}{\epsilon} \log n)$ space.

The algorithm in retrospect. A metaphoric view of the algorithm could be the following: Consider the DP table generated by the optimum algorithm with n columns and B rows, the bottommost row corresponding to $\textsc{Terr}[i, 1]$. This new algorithm maintains a "front" that moves from left to right and creates (approximately) the same table as the optimal algorithm, but only chooses to remember a few "highlights" (see Figure 84.1). The highlights correspond to boundary points that are sufficient to construct an approximate histogram. We can view the optimum algorithm as using n buckets to represent the nondecreasing error function $\textsc{Terr}[i, k-1]$ exactly. But we need at most $O(\frac{1}{\delta} \log n)$ buckets for polynomially bounded input if we approximate the function in the above geometrically growing fashion. This geometrically growing staircase has been subsequently used in several problems of interest in time windowed data streams [4].

Improving the above algorithm. The algorithm mentioned above still computed all the $\theta(nB)$ table entries, though each entry was computed faster. We were forced to evaluate all entries in the table in the absence of any indication whether that value will be irrelevant later in a streaming setting.

To improve the algorithm, we begin by ignoring the streaming aspect and develop on an offline algorithm with $O(n)$ memory. We subsequently show how to adapt the improved algorithm to a stream setting. One way of viewing the new offline algorithm is that we want to create a similar dynamic table as the optimal, but we only want to compute the $\text{APXERR}[j, k-1]$ entries that are useful for some $\text{APXERR}[i, k]$ entry. *In a sense, we want the front to move from bottom to top and only evaluate the necessary values* (see Figure 84.1), and this is requires the offline setting.

Note, we immediately have a problem that $\text{APXERR}[i, k+1]$ may depend on $\text{APXERR}[i-1, k]$ and $\text{APXERR}[i-1, k]$ was later replaced by some $\text{APXERR}[i', k]$ where $i' > i$. If we are only computing the values that are necessary, we will be computing different values since $\text{APXERR}[i, k+1]$ now has to use $\text{APXERR}[i', k]$. This is where the induction in the proof of the earlier algorithm fails. However, the reassuring aspect is that $\text{APXERR}[i', k]$ must have been within a $(1+\delta)$ factor of $\text{APXERR}[i-1, k]$ and a more subtle induction goes through. This idea in itself gives an algorithm with running time $O(n + \frac{B^3 \log n}{\epsilon^{-2}})$. But we will improve the algorithm even more.

Note that we are interested in the entry $\text{APXERR}[B, n-1]$. This is the top right-hand corner of the table. Now, the elements in the bottom right-hand corner of the table are likely to contain very large values, because they correspond to the approximation by very few buckets. Likewise, the elements in the top left-hand corner are likely to contain very small answers, which correspond to approximating very small amounts of data with a large number of buckets. Either of these sets of values is unlikely to influence the optimum solution. However, we need to quantify "large" and "small" in this discussion. The idea would be to first find the *scale of the optimum solution* and subsequently search in that scale to find the (near) best solution.

Assume that we were guaranteed that the optimum solution is less than 2Δ for some Δ. We find the largest i such that $\text{APXERR}[i, 1] = \text{SQERROR}(1, i) \leq (1+\epsilon)2\Delta$. This we set to be b_u. We proceed backward to determine the smallest number a_u such that $\text{APXERR}[a_u, 1] + \frac{\epsilon\Delta}{B-1} \leq \text{APXERR}[b_u, 1]$. This defines the last interval (a_u, b_u). We set $b_{u-1} = a_u - 1$ and proceed (backward). After we have created the list of intervals corresponding to 1 bucket (or k buckets), we will proceed to the list of intervals corresponding to 2 (or $k+1$) buckets. The size of each list will be $O(\frac{B}{\epsilon})$. Finding the smallest a_u would involve a binary search and each evaluation of $\text{APXERR}[i, k]$ would involve using $O(\frac{B}{\epsilon})$ values from the list corresponding to b_u for $k-1$ buckets. The running time over all the B lists can be shown to be $O(\frac{B^3}{\epsilon^2} \log n)$. Observe that the algorithm incurs approximation error additively and over the entire algorithm, the total error can be shown to be an additive $\epsilon\Delta$.

Suppose we started from the smallest possible nonzero value as Δ and we got a solution whose error is more than $2\Delta(1+\epsilon')$. Then we know that the optimum solution is above 2Δ and we can double the estimate of Δ. This way, after at most $O(\log n)$ rounds we will get to a point where we get a solution whose error is at most $2\Delta(1+\epsilon')$, but recursively we have maintained the invariant that the optimum is at least Δ. At this point, set $\Delta' = \Delta(1+\epsilon')$; and by virtue of the existence of some solution of cost $2\Delta(1+\epsilon')$, we are guaranteed that the optimum is within $2\Delta'$. We choose a ϵ'' such that $\epsilon'' = \epsilon\Delta/\Delta'$ and apply the above algorithm. The final solution has additive error $\epsilon\Delta$, which is a $(1+\epsilon)$ approximation since Δ is less than the optimum solution. The running time of the algorithm is $O(\frac{B^3 \log n}{\epsilon^2} + \frac{B^3 \log n}{\epsilon^2} \log n)$ considering all $O(\log n)$ rounds. Now observe that we can set $\epsilon' = 1$, i.e., try to get a fast 4 approximation. We compute the sums $\sum_i X(i)$ etc., in $O(n)$ additional time. In summary,

Theorem 84.2

We can find a $(1+\epsilon)$ approximation to the optimal histogram in $O(n + B^3 \log^2 n + \frac{B^3}{\epsilon^2} \log n)$ time.

A return to streams. The above algorithm appears to be hopelessly offline. In our metaphor of "front," we are proceeding row by row upward, and the entire data are required to be present to allow us to compute SQERROR(). The idea we would now use is to read in *a block of data* of size M in left-to-right order, but use the bottom-to-top approach to construct the staircases for the new data (using staircases of the old data). We would have to maintain the geometric approximation as in the first-approximation algorithm (since across different blocks we cannot proceed backward as in the second algorithm). The number of items we would consider for each list would be $O(\frac{B}{\epsilon}\log n)$ as in the first algorithm, plus $\frac{n}{M}$, corresponding to the endpoints of the blocks since in each block we will proceed backward and always evaluate the endpoint. To find the smallest a_u, however, would only require $O(\log M)$ evaluations since we would be searching over the new elements only. Thus, over all the lists the algorithm will use $O(B(\frac{n}{M}+\frac{B}{\epsilon}\log n)\log M)$ evaluations of some APXERR$[i,\cdot]$. We will use a better algorithm to compute APXERR$[i,k]$. Along with the lists $Q[k]$ of intervals, where APXERR$[\cdot,k]$ increase geometrically in powers of $(1+\frac{\epsilon}{2B})$, we would keep track of a sublist $SubQ[k]$ inside this list of intervals where the APXERR$[\cdot,k]$ increase in powers of 2. Thus, the sublist will be of size $O(\log n)$. We will use $SubQ[k-1]$ to compute a 4 approximation (say A) of TERR$[i,k]$ of APXERR$[i,k]$. Then we would proceed backward inside the list $Q[k-1]$ and only consider the APXERR$[j,k-1]$ elements that are separated by $A/(cB)$ for some constant c, and *use these new elements* to compute a better approximation of TERR$[i,k]$. This approximation can be shown to be $(1+\delta)^k$; the proof is detailed and is omitted. The upshot of this more complicated algorithm is that the time to compute each APXERR$[i,k]$ is $O(\log n+\frac{B}{\epsilon}\log\tau)$ (where $\tau=\frac{B\log n}{\epsilon}$, the same as earlier); the extra log term arises from the backward binary search inside $Q[k-1]$. Thus, the total running time (we add the $O(n)$ time to compute the sums for all the elements) is

$$n+\left(\log n+\frac{B}{\epsilon}\log\tau\right)B\left(\frac{n}{M}+\frac{B}{\epsilon}\log n\right)\log M$$

We can now set M to get the coefficient of n to be a true constant, and thus

Theorem 84.3

Let $\tau=\frac{B}{\epsilon}\log n$ and $M=O((\frac{B}{\epsilon}\log\tau+\log n)B\log\tau)$. We can construct a $(1+\epsilon)$ approximation data stream algorithm for computing the optimal histogram that runs in time $O(n+M\tau)$ and uses space $O(B\tau+M)$.

For fixed B, ϵ, and any $\gamma>0$, using $O(\gamma\log n)$ space we get a $(1+\epsilon)$ approximation in $O(n+\frac{n\log\log n}{\gamma})$ time, which is a nice tradeoff. A natural question that would arises at this point—do these approximations help? Note that the approximations were motivated from very common sense "pruning strategies" or heuristics that should be a part of a good code. It is gratifying that in this case we can analyze the pruning strategies and in fact prove their correctness as well as improved performance. For an implementation, the dependence on ϵ, B matters and getting a better theoretical algorithm does allow us to have better algorithms for practice.

84.3.2 Beyond ℓ_2^2 Error: Workloads, Piecewise Polynomials

If we inspect the algorithms in the previous section, the following ideas were used in the DP and the approximation algorithm(s), respectively:

1. *OPT.* The error SQERROR(i,j) of a bucket depends only on the values in the bucket and the endpoints i, j. We can maintain small information for each element such that given any i, j the value of SQERROR$(i+1,j)$ can be computed efficiently. The overall error is the sum of the errors of the buckets.
2. *APX.* The error is *interval monotone*, i.e., a subinterval has error no more than the whole interval. The minimum nonzero error and the maximum error are lower and upper bounded, respectively, by polynomials in n.

We can now revisit the proof in the previous section (with $P = Q = T = O(1)$), and the next theorem follows:

Theorem 84.4

Suppose that we are given a histogram construction problem where the error $E_T[\cdot, \cdot]$ satisfies the above conditions. Suppose that the error of a single bucket $E_B(i+1, j)$ can be computed in time $O(Q)$ from the records INFO$[i]$ *and* INFO$[j]$ *each requiring $O(P)$ space. Assume that the time to create the $O(P)$ structure is $O(T)$; then by changing the function that computes the error given the endpoints we achieve the following:*

- *(i) We can find the optimum histogram in time $O(nT + n^2(B + Q))$ time and $O(n(P + B))$ space.*
- *(ii) In $O(nT + QB^3(\log n + \epsilon^{-2}) \log n)$ time and $O(nP)$ space, we can find a $(1 + \epsilon)$ approximation to the optimum histogram.*
- *(iii) In $O(nT + M_Q\tau)$ time and $O(PB\tau + M_Q)$ space we can find a $(1+\epsilon)$ approximation to the optimum histogram over a data stream where $M_Q = B(\frac{QB}{\epsilon} + Q \log n + \frac{B}{\epsilon} \log \tau) \log(Q\tau)$ and $\tau = B\epsilon^{-1} \log n$.*

Example 84.1 **(Workloads)**

Workloads are weighted ℓ_p norms. Typically, the workload is specified as a k-bucket histogram as well (since specifying n weights requires a lot of space) and each $\mathrm{E}_B()$ can be computed in time $Q = O(k)$ time. The space requirement to store the lists increases by an additive $O(k)$ since it is simpler to add the endpoints of the workload histogram to all the queues. $T = P = O(1)$ in this case.

Example 84.2 **(Piecewise Polynomials)**

For polynomials of degree d, we need to store prefix sums such as $\sum_r X(r)^m$ for $0 \leq 2d + 2$. To find the best representative, we need to solve an $O(d) \times O(d)$ matrix that makes $Q = O(d^3)$. $P = T = O(d)$ in this case.

Example 84.3 **(ℓ_1 Error)**

In this case, the representative of a bucket is the median. In an offline setting, we can preprocess the data in $O(n \log n)$ time and space to achieve $Q = O(\log^2 n)$. In the stream setting, we need to prove that an approximate median of rank within $\frac{n}{2} \pm \epsilon n$ increases the error of a bucket by $(1 + \epsilon)$ factor. Approximate medians can be found using the algorithms of Manku et al. [5] or Greenwald and Khanna [6] using $O(\frac{\log n}{\epsilon})$ space. However, this needs to be done for each of the $B\tau$ endpoints (each of which could potentially form a bucket with the current data we are seeing). Thus, we can apply Theorem 84.4 (iii) with $T = B\tau \log \frac{\log n}{\epsilon}$, $Q = P = \log \frac{\log n}{\epsilon}$.

Example 84.4 **(χ^2 Error and Information Distances)**

In this case, the error of representing a set of numbers $v_1, \ldots, v_m$ by h is given by $\sum_r \frac{(v_r - h)^2}{h}$. Using prefix sums similar to ℓ_2^2, we can show $Q = T = P = O(1)$. This is interesting since one of the objectives of histograms is to represent distributions and information theoretic metrics are obviously more suited for comparing distributions.

Example 84.5 **(Relative Error)**

One of the issues with ℓ_p error is that approximating 1000 by 1010 counts as much toward the error as approximating 1 by 11. One way of ameliorating the problem is to define a relative error measure that computes a function (ℓ_2, ℓ_1 norm of the vector) $\frac{|X(i) - \hat{X}(i)|}{\max\{|X(i)|, c\}}$, where $\hat{X}(i)$ is the estimate of $X(i)$ constructed from the synopsis. c is a constant to avoid division by 0 and the effect of arbitrarily small numbers. The different measures lead to different settings of P, Q, and T. For example, for relative ℓ_2 we need to compute the harmonic mean, but that can be achieved with $P = Q = T = O(1)$.

84.3.3 ℓ_∞ and Variants

The histogram algorithms can be significantly simplified for ℓ_∞ variants (workload, relative error, etc.). Note that

Observation 84.3

For most reasonable error metrics based on ℓ_∞ (relative ℓ_∞), the error of a bucket depends on the (suitably weighted) maximum and minimum values in the bucket.

Thus, as long as the maximum and minimum values are fixed the error of a bucket does not change. If we were told that the error of an interval using k buckets is τ, we can verify the claim by "eating up" maximal subintervals from the left of error τ and see if we can "cover" the entire interval. Assuming that we can compute the error of any interval in $O(Q)$ time, the running time of such an algorithm is $O(kQ \log n)$ using binary search. We can easily maintain a tree using $O(n)$ preprocessing and $O(n)$ space that gives gives $Q = \log n$. Let the error of a single bucket defined by the interval $[a, b]$ be $\mathrm{E}_{B,\infty}(a, b)$.

The above gives a simple algorithm where we guess the *first* bucket. If the first bucket is defined by the interval $[0, i]$, then we can check if $B - 1$ buckets cover the interval $[i + 1, n]$ using $\tau = \mathrm{E}_{B,\infty}(0, i)$. If yes, then we need to try lower values of i (or i may be the correct answer). Otherwise, we know that the error of the optimum solution is larger than τ. Thus, either (a) we need to increase i, which increases τ or (b) we need to increase τ but $[1, i]$ is the first bucket. So we find an i such that if the error of the optimum solution is τ, then it satisfies $\mathrm{E}_{B,\infty}(0, i) < \tau \leq \mathrm{E}_{B,\infty}(0, i + 1)$. This is found in time $O(B \log^3 n)$ using binary search. Now if the optimum error of representing the interval $[i + 1, n]$ using $B - 1$ buckets is τ^* (which we will find recursively), then the final error is $\min\{\mathrm{E}_{B,\infty}(0, i + 1), \tau^*\}$. More explicitly, if $\tau^* < \mathrm{E}_{B,\infty}(0, i+1)$ then the optimum solution is the solution of $[i+1, n]$ (found recursively) along with the first bucket defined by the interval $[0, i]$. Otherwise, the solution is the result of taking out intervals whose error is less or equal to $\mathrm{E}_{B,\infty}(0, i+1)$. Note that this sets up a recursion $f(B) = B \log^3 n + f(B-1)$ and thus we conclude:

Theorem 84.5

For variants of ℓ_∞ error (weighted, workload, relative error), we can compute the optimal histogram in time $O(n + B^2 \log^3 n)$ and $O(n)$ space.

However, in a streaming scenario where we cannot afford linear space, we can use an algorithm similar to (iii) in Theorem 84.4. We can begin by writing a *worse* algorithm, which is $O(n^2 B)$ time but computes the optimum solution in a fashion similar to ℓ_2^2, but uses $\mathrm{E}_{T,\infty}[i, k] = \min_j \max\{\mathrm{E}_{T,\infty}[j, k - 1], \mathrm{E}_{B,\infty}(j + 1, i)\}$. But then,

Observation 84.4

We can compute the minimum of $\max\{f(j), g(j)\}$ *in $O(\log n)$ evaluations of $g()$ if $f(j) = E_{T,\infty}[j, k-1]$ is nondecreasing and $g(j) = E_{B,\infty}(j + 1, i)$ is nonincreasing.*

Therefore, we immediately improve the worse optimum algorithm to run in time $O(nB \log^2 n)$ using the $O(n)$ preprocessing to answer $\mathrm{E}_{B,\infty}()$ in $O(\log n)$ time. Now consider maintaining a staircase approximating $\mathrm{E}_{T,\infty}[j, k - 1]$ using $\tau = O(\frac{B \log n}{\epsilon})$ endpoints. Now, the block-by-block algorithm performs $\frac{n}{M} + \tau$ insertions into each interval list. Each requires a binary search of $\log M$ and over B lists we evaluate $\textsc{ApxE}_\infty[]$ at most $O(B(\frac{n}{M} + \tau) \log M)$ times, each requiring $O(\log \tau)$ time. There is one complication, namely, in evaluating $\mathrm{E}_{B,\infty}(b_u, i)$ if b_u was in some block $r - 2$ or before and i was in block r. The answer depends on the maximum and minimum values in block $r - 1$. So as we process one block and move to the next, we may have to update the $O(B\tau)$ entries in the list to take care of the above issue. This adds $O(\frac{n}{M} B\tau)$ to the running time. The overall running time is thus

$$B\left(\frac{n}{M} + \tau\right)(\log M)(\log \tau) + \frac{n}{M} B\tau + n$$

We can set $M = O(B\tau)$ to make the coefficient of n to be $O(1)$ and thus

Theorem 84.6

We can compute a $(1 + \epsilon)$ approximation of the ℓ_∞ variants (workloads, weighted, etc.) in $O(n + \frac{B^2 \log n}{\epsilon} \log^2 \frac{B \log n}{\epsilon})$ time and $O(\frac{B^2 \log n}{\epsilon})$ space over a data stream.

84.4 Wavelet Synopses

We begin with a brief review of wavelets and their main properties. One of the most important reasons for the popularity of wavelets is captured in Proposition 84.1, which states that (for compact wavelets) at most $O(\log n)$ basis vectors are relevant to a point. Also, the fact that the basis set is orthonormal and the existence of fast forward and inverse transforms has been a strong attraction for their wide use.

84.4.1 A Compact Primer on Compact Wavelets

Definition 84.2

The support of any vector Ψ is the set $\textsc{Supp}(\Psi) = \{t|\Psi(t) \neq 0\}$.

Definition 84.3

Let $h[]$, $g[]$ be two arrays defined on $\{0, 1, \ldots, 2q-1\}$, s.t., $g[k] = (-1)^k h[2q-1-k]$. Assume that $\sum_k h[k] = \sqrt{2}$ and $\sum_k g[k] = 0$, (along with few other properties, see Refs. [7,8]). Let $\phi_{0,s}(t) = \delta_{st} \in \mathcal{R}^n$, i.e., the vector which is 1 at s and 0 everywhere else. Define $\phi_{j+1,s} = \sum_t h[t-2s]\phi_{j,t}$ and $\psi_{j+1,s} = \sum_t g[t-2s]\phi_{j,t}$.

The set of wavelet vectors $\{\psi_{j,s}\}_{(j,s)\in\mathbb{Z}^2}$ define an orthonormal basis for $\mathcal{R}^n$. For ease of notation, we will use both ψ_i and $\psi_{j,s}$ depending on the context and assume that there is a consistent map between them. The function $\psi_{j,s}$ is said to be centered at $2^j s$ and of scale j and is defined on at most $(2q-1)2^j$ points. It can be shown that $\phi_{j,s} = \sum_t h[s-2t]\phi_{j+1,t} + \sum_t g[s-2t]\psi_{j+1,t}$, [7]. Further $\phi_{j,s}(x)$, $\psi_{j,s}(x)$ when scaled to the (continuous) domain $[0, 2q-1]$ converge to $2^{-j/2}\phi(\frac{x-2^j s}{2^j})$,$2^{-j/2}\psi(\frac{x-2^j s}{2^j})$; i.e., the vectors look similar, but are shifted and scaled.

Example 84.6 (Haar Wavelets)

In this case, $q = 1$ and $h[] = \{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\}$. Thus $g[] = \{\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\}$. Given X, the algorithm to compute the transform finds the "difference" coefficients $d_1[i] = \frac{X(2i)-X(2i+1)}{\sqrt{2}}$. The "averages" $\frac{X(2i)+X(2i+1)}{\sqrt{2}}$, correspond to $a_1[i]$, and the entire process is repeated on these $a_1[i]$ but with $n := n/2$ since we have halved the number of values. In the inverse transform we get, for example, $a_0[0] = (a_1[0] + d_1[0])/\sqrt{2} = X(0)$ as expected. The coefficients naturally defines a coefficient tree where the root is $a_{\log n+1}[0]$ (the overall average scaled by $\sqrt{n}$) with a single child $d_{\log n}[0]$ (the scaled differences of the averages of the left and right halves). Underneath $d_{\log n}[0]$ lies a complete binary tree as shown in Figure 84.2(c). Note that in this case the support of the basis vectors defines a hierarchical structure and each wavelet coefficient "affects" the values in its subtree only.

Note that ψ is discontinuous, i.e., if a wavelet basis vector with a large support is mapped to the continuous interval $[0, 1]$, the transition from positive to negative values remain and the gap is $\tilde{}2^{j/2}$ at scale j. Thus, the synopses using Haar wavelets are better suited to handle "jumps" or discontinuities in data. This simple wavelet proposed in 1910 is still useful since it is excellent in concentrating the *energy* of the transformed signal (sum of squares of coefficients). A natural question arises if "smooth" wavelets exist, i.e., when a wavelet vector with a large support is mapped to the continuous interval $[0, 1]$, the values get significantly closer. The seminal work of Daubechies gives us several examples which we discuss next ([8]).

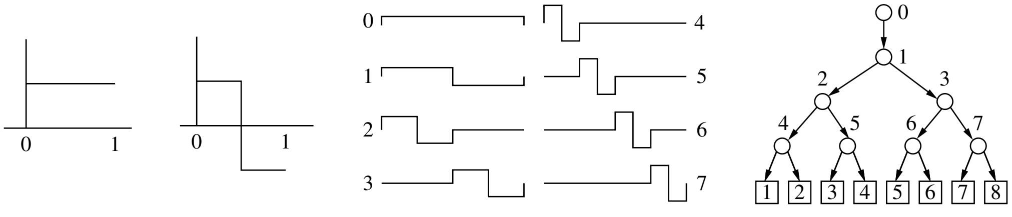


FIGURE 84.2 The ϕ, ψ, the set of Haar basis vectors at scale 3, and the tree defined by the coefficients.

Example 84.7 (Daubechies Wavelets D_2)

In this case, $q = 2$ and $h[] = \{\frac{1+\sqrt{3}}{4\sqrt{2}}, \frac{3+\sqrt{3}}{4\sqrt{2}}, \frac{3-\sqrt{3}}{4\sqrt{2}}, \frac{1-\sqrt{3}}{4\sqrt{2}}\}$. Thus $g[] = \{h[3], -h[2], h[1], -h[0]\}$. The ϕ and the ψ functions are shown below (normalized to the domain [0,1]) and they converge quite rapidly. The coefficients now form a graph rather than a tree, which is given in Figure 84.2. The D_q wavelets have compact support (q is a fixed integer) but are unfortunately asymmetric. It turns out that Haar wavelets are the unique real symmetric compactly supported wavelets [8]. Moving to the complex domain one can define symmetric bi-orthogonal wavelets.

Proposition 84.1

For a compactly supported wavelet, there are $O(q \log n)$ basis vectors with a nonzero value at any point t. Further, given t, $\psi_{j,s}(t)$ and $\phi_{j,s}(t)$ can be computed in $O(q \log n)$ time.

The Cascade algorithm for $\langle X, \psi_{j,s}\rangle$, $\langle X, \phi_{j,s}\rangle$. To compute the **forward transform**: Given a function X, set $a_0[i] = X(i)$, repeatedly compute $a_{j+1}[t] = \sum_s h[s-2t]a_j[s]$ and $d_{j+1}[t] = \sum_s g[s-2t]a_j[s]$. It is easy to see that $a_j[t] = \langle X, \phi_{j,t}\rangle$ and $d_j[t] = \langle X, \psi_{j,t}\rangle$. To compute the **inverse transform**, we compute $a_j[t] = \sum_s h[t-2s]a_{j+1}[s] + \sum_s g[t-2s]d_{j+1}[s]$.

Definition 84.4

Let $\mathcal{W}(X)$ denote the wavelet transform, i.e., $\mathcal{W}(X)(t) = \langle X, \psi_t\rangle$ and let $\mathcal{W}^{-1}(Z) = \sum_i Z(i)\psi_i$ denote the inverse transform.

Recall that the synopsis problem is: Given a wavelet basis $\{\psi_i\}$ and $X = X(0), \ldots, X(n-1)$ in a data stream, find a set of values $\{Z(i)\}$ with at most B nonzero values minimizing a suitable function of $X - \sum_i Z(i)\psi_i$.

84.4.2 Wavelet Synopses and ℓ_2 Theory

Suppose that we were interested in minimizing $\|X - \mathcal{W}^{-1}(Z)\|_2$. We can use the result of Parseval which states that "lengths are preserved under rotations." Since an orthonormal transformation defines a rotation and the wavelet basis vectors define an orthonormal basis, $\|X - \mathcal{W}^{-1}(Z)\|_2 = \|\mathcal{W}(X - \mathcal{W}^{-1}(Z))\|_2$. Since the transformation is linear we get $\|X - \mathcal{W}^{-1}(Z)\|_2 = \|\mathcal{W}(X) - \mathcal{W}(\mathcal{W}^{-1}(Z))\|_2$, which is equivalent to minimizing $\sum_i (Z(i) - \mathcal{W}(X)(i))^2$. The constraint is that at most B of the $Z(i)$s can be nonzero. The solution is clearly choosing the largest $|\mathcal{W}(X)(i)| = |\langle\psi_i, X\rangle|$ and set $Z(i) = \mathcal{W}(X)(i) = \langle\psi_i, X\rangle$. *Observe that the fact that we retain some of the coefficients was a consequence of the proof and not a constraint.*

The synopsis construction problem for ℓ_2 error reduces to choosing the largest (ignoring sign) wavelet coefficients of the data. It is not too difficult to see that this can be computed over a data stream. The simplest way of viewing the computation is a "level-by-level" construction of running several algorithms in parallel, each corresponding to a level. The basic insight of the paradigm is reduce-merge [9] and for streaming algorithms this idea was first used in the context of clustering [10]. We need to implement the cascade algorithm in a similar format.

We describe an algorithm that reads a stream of values $X(0), \ldots, X(n-1)$ and outputs the set of coefficients $\langle\psi_i, X\rangle$ (in some order). This algorithm uses $O(q \log n)$ space. We can feed the output of this algorithm that maintains the largest (ignoring signs) B values seen in the stream. This can be achieved using $O(B)$ space in $O(n)$ time.

In the lowest level, the algorithm sees the data and computes $d_1[q]$ for the first $2q$ values. For the Haar basis, this is $\frac{X(0)-X(1)}{\sqrt{2}}$. The value $a_1[]$ (for Haar, $a_1[0] = \frac{X(0)+X(1)}{\sqrt{2}}$) is passed to the algorithm in the next (higher) level.

The algorithm in the lowest level now proceeds to read two new values and output $d_1[q+1]$. For a non-Haar basis, there is an issue of wrap-around and the first $2q-2$ data values are useful for a coefficient that depends on data that arrive at the end; thus they need to be stored. For Haar, the values $X(0)$, $X(1)$ can be discarded. It is clear that in each level j we need to store $O(q)$ information and output the coefficients of scale j. The total space of the algorithm across all levels is $O(q \log n)$.

Theorem 84.7

We can compute the optimal wavelet synopsis under ℓ_2 error using $O(n)$ space and $O(B + q \log n)$ space, for any compact wavelet basis.

84.4.3 Wavelet Synopses under Non-ℓ_2 Error

Suppose that we were interested in minimizing $\| X - \mathcal{W}^{-1}(Z)\|_\infty$. We cannot use the result of Parseval since the ℓ_∞ norm is not preserved under rotations. In fact, we can easily see that storing any subset wavelet coefficient is suboptimal. Consider $B = 1$ the Haar basis and $X = \{2, 2, 2, 0\}$, then $\mathcal{W}(X) = \{3, 1, 0, \sqrt{2}\}$. The best solution restricted to storing the coefficients is $Z = \{3, 0, 0, 0\}$ and $\mathcal{W}^{-1}(Z) = \{\frac{3}{2}, \frac{3}{2}, \frac{3}{2}, \frac{3}{2}\}$ with $\| X - \mathcal{W}^{-1}(Z)\|_\infty = 1.5$. It is easy to see that $Z = \{2, 0, 0, 0\}$ gives $\| X - \mathcal{W}^{-1}(Z)\|_\infty = 1$. It is not difficult to construct a similar example for ℓ_1 error as well. We will prove a polynomial time approximation scheme for the Haar basis, which extends to a quasi-polynomial time scheme for a general compact basis. We begin by computing a lower bound for any ℓ_p error.

Let the minimum possible value of $\| X - \mathcal{W}^{-1}(Z)\|_p = \tau_{opt}$ be achieved by the solution Z^*. For all j, we have $-\tau_{opt} \le X(j) - \mathcal{W}^{-1}(Z^*)(j) \le \tau_{opt}$. Multiplying the equation by $\psi_i(j)$ and summing over j, we get $-\|\psi_i\|_1 |\tau_{opt}| \le \langle X, \psi_i\rangle - \langle \psi_i, \mathcal{W}^{-1}(Z^*)\rangle \le \|\psi_i\|_1 |\tau_{opt}|$. But $\langle \psi_i, \mathcal{W}^{-1}(Z^*)\rangle = z_i^*$. Thus, we can write a system of equations

$$\begin{aligned} &\min \tau \qquad\qquad\qquad \text{s.t.} \\ &-\tau \|\psi_1\|_1 \le \langle X, \psi_i\rangle - z_1^* \le \tau \|\psi_1\|_1 \ \text{ for all } i \\ &\text{At most } B \text{ of the } z_i^* \text{ are nonzero} \end{aligned} \tag{84.1}$$

The constraints are satisfied by τ_{opt}. Let the optimum solution of the above system be τ^*. Thus $\tau^* \le \tau_{opt}$. The system of equations is *nonlinear*.[1] However, the above system (84.1) can be solved optimally, and the minimum solution is the $(B + 1)$th largest (ignoring signs) value of $\langle X, \psi_i\rangle / \|\psi_i\|_1$. We can also derive the following:

$$-\|\psi_i\|_1 |\tau_{opt}| \le \langle X, \phi_i\rangle - \langle \phi_i, \mathcal{W}^{-1}(Z^*)\rangle \le \|\psi_i\|_1 |\tau_{opt}|$$

The above equation shows the effect of the coefficients whose support contains the entire support of ψ_i. In effect, given the optimum error, we have a handle on how the rest of the input must behave in relation to the $\{X(j) \mid j \in \textsc{Supp}(\psi_i)\}$. We are interested in keeping track of all possible scenarios of $\langle \phi_i, \mathcal{W}^{-1}(Z^*)\rangle$.

84.4.3.1 Streaming PTAS for Haar Systems

We solve the problem in a scaled bases and translate the solution to the original basis. We begin with the following:

Proposition 84.2

Define $\psi^{\mathcal{P}}_{j,s} = 2^{-j/2}\psi_{j,s}$ and $\psi^{\mathcal{D}}_{j,s} = 2^{j/2}\psi_{j,s}$. Likewise, define $\phi^{\mathcal{P}}_i$, $\phi^{\mathcal{D}}_i$. The Cascade algorithm used with $\frac{1}{\sqrt{2}}h[]$ computes $\langle X, \psi^{\mathcal{P}}_i\rangle$ and $\langle X, \phi^{\mathcal{P}}_i\rangle$. The problem of finding a synopsis Z with basis $\{\psi_i\}$ is equivalent to finding a synopsis Y using the basis $\{\psi^{\mathcal{D}}_i\}$ for the inverse transform, i.e., we are seeking to minimize a function of $X - \sum_i Y(i)\psi^{\mathcal{D}}_i$. The correspondence is $Y(i) = 2^{-j/2} Z(i)$ where $i = (j, s)$.

[1] The last constraint can be expressed as a linear constraint, but it is not clear how to "round" the fractional solution we would obtain.

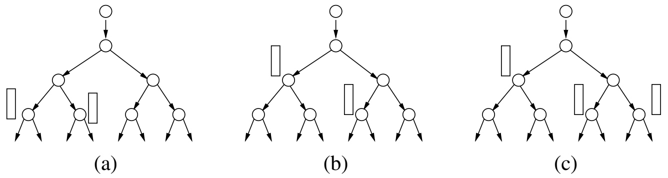


FIGURE 84.3 The dynamic programming along the stream.

Lemma 84.1

Let Y^* be the optimal solution using the basis set $\{\psi_i^{\mathcal{P}}\}$ for the reconstruction, i.e., $\hat{X} = \sum_i Y^*(i)\psi_i^{\mathcal{P}}$ and $\|X - \hat{X}\|_p = \tau_{opt}$. Let Y^ρ be the vector where each $Y^*(i)$ is rounded to the nearest multiple of ρ. If $X^\rho = \sum_i Y^\rho(i)\psi_i^{\mathcal{P}}$ then $\|X - X^\rho\|_p \leq \tau_{opt} + O(qn^{1/p}\rho \log n)$.

The proof follows from standard accounting of the error at each point and the triangle inequality. If we can find Y^ρ for $\rho = \frac{\epsilon\tau_{opt}}{qn^{1/p}\log n}$ we have a $(1+\epsilon)$-approximation. Note that $q = 1$.

Haar and the art of streaming We can find Y^* using a DP. At each node i, we will compute a table $\text{Err}_i[v, b]$ that corresponds to the contribution of the subtree rooted at i to the error of the minimum solution assuming that the combined effect (signed sum) of all coefficients corresponding to ancestors of i is v. In other words, this is the interaction between the subtree rooted at i and the rest of the tree. Note that this value will lie in the range $\langle X, \phi_i^{\mathcal{P}}\rangle \pm \tau^*$. The size of the table is $O(\frac{\tau_{opt}B^2}{\rho})$, the extra B term arises from keeping track of the solution.

We can compute the DP table at i given the tables for its children. As we read data from left to right, we can keep generating the tables in postorder fashion. Figure 84.3(a) shows the state when we have read four values and have created two tables for two sibling nodes. We can discard the tables for the children once we have computed the table at their parent. This is shown in Figure 84.3(b), along with the evolution of the state corresponding to more data being read. When we reach the configuration in Figure 84.3(c), we would recursively collapse the tables and propagate them upward and finish the computation.

It is immediate that the space taken would be $\log n$ times the space of a table. For each entry, we have to decide on the value of $Z(i)$ and the allocation of b coefficients to the subtrees rooted at its children. This corresponds to $O(\frac{\tau_{opt}B}{\rho})$ choices, but for nodes with very few descendants this can be reduced. However, there is small complication: we cannot solve the system (84.1) since the order of the coefficients cannot be determined until we have seen all the data. To avoid this, we will guess the value of τ_{opt} and verify that no more than B coefficients exceed the required limit. This would involve a further $O(\log n)$ term (assuming polynomially bounded input). Thus, we can conclude with:

Theorem 84.8

We can compute the optimal wavelet synopsis under ℓ_p error using $O(n^{1+\frac{2}{p}} B^2 \frac{\log^3 n}{\epsilon^2})$ time and $O(\frac{B^2 n^{\frac{1}{p}} \log^3 n}{\epsilon})$ space, for Haar wavelets.

The running time can be improved slightly since at the lower part of the tree, the number of descendants is less than B. The above results extend to multiple dimensions. In the case of non-Haar compact systems, e.g., Daubechies-4, we can achieve a quasi-polynomial time approximation scheme.

84.4.4 Restricted Optimizations

A question arises in this context—"If we are only interested in a synopsis such that $Z(i) = 0$ or $Z(i) = \langle\psi_i, X\rangle$, what is the best possible reconstruction?" One motivation of the question can be that we may be interested in multiple uses of the synopsis and retaining the wavelet coefficients presses some advantage in some other direction. The above problem can be solved optimally in $O(n^2 \log B)$ time and $O(n)$ space for a variety of error measures (any weighted ℓ_p, which subsumes all "workload" versions, for ℓ_∞ the time is $O(n^2)$) for the Haar system.

Problem 84.5

What is the relationship between the minimum error of the restricted and the original problem?

We show that a modified greedy heuristic that retains B coefficients of the data gives a $O(n^{\frac{1}{p}} \log n)$ approximation algorithm in $O(n)$ time and $O(B + \log n)$ space for any compact wavelet. Recall the equation system (84.1). Consider the set of values $\{\frac{|\langle X, \psi_{i_j} \rangle|}{\|\psi_{i_j}\|_1}\}$ in the decreasing order. Set $Z(i_j) = \langle X, \psi_{i_j} \rangle$ for $1 \leq j \leq B$ and $Z(i) = 0$ otherwise. The resulting Z is an optimal solution of the system (84.1), but not of the original synopsis construction problem. To analyze the error we would require the next lemma, which follows from the properties of compact wavelets.

Lemma 84.2

For all basis vectors ψ_i, there exists a constant C (depends on q) such that $\|\psi_i\|_\infty \|\psi_i\|_1 \leq C$.

The above proposition allows us to conclude $\|X - \mathcal{W}^{-1}(Z)\|_p = O(\tau n^{\frac{1}{p}} \log n)$ where $\tau \leq E$. Thus,

Theorem 84.9

The algorithm of retaining the B coefficients with largest $\frac{|\langle X, \psi_i \rangle|}{\|\psi_i\|_1}$ is an $O(n^{1/p} \log n)$ approximation for the synopsis construction problem under the ℓ_p norm. The algorithm can be implemented over a stream to run in $O(n)$ time and $O(B + \log n)$ space. This is a simultaneous approximation of all ℓ_p norms.

84.5 From (Haar) Wavelets Form Histograms

The first idea that comes to one's mind is that can we simply represent X using a number (to be chosen later) of large projections onto the wavelet basis, i.e., using a few large wavelet coefficients? The idea does work, but requires many wavelet coefficients. Stepping back, we can reason that what we want to achieve from the process is the discovery of the places where X "jumps" significantly and use that information. This points us to the Haar basis, since the Haar basis is best suitable for functions with discontinuities (jumps). The idea would be to find a few large coefficients and store information relevant to the *boundaries* defined by the function. Note that storing information about the boundaries is tantamount to storing more coefficients, except that we use less space. As mentioned, we will focus on the Haar system. Define $\textsc{Proj}(X, \mathcal{V})$, to be the projection of X onto the basis vectors in set $\mathcal{V}$.

Definition 84.5

Given a B-bucket histogram H, define $\textsc{Poss}(H)$, *the set of wavelet vectors ψ_i such that the support* $\textsc{Supp}(\psi_i)$ *is* **not** *completely contained inside one of the buckets. Observe that if $\psi_i \notin \textsc{Poss}(H)$ then $\psi_i \cdot H = 0$.*

Note that $|\textsc{Poss}(H)| \leq 2B \log n$ and $\textsc{Poss}(H)$ is hierarchically closed (upward) with respect to the coefficient tree, since the coefficients form a tree defined by the set $\textsc{Supp}()$.

Lemma 84.3

Given a B-bucket histogram H, let $\mathcal{V}_0$ be the set of $2B(1 + \epsilon^{-2}) \log n$ basis vectors that have the largest (unsigned) projection with X. Let $\mathcal{V}$ be the hierarchical upward closure of $\mathcal{V}_0$. Then $\|\textsc{Proj}(X, \textsc{Poss}(H) \setminus \mathcal{V})\|_2^2 \leq \epsilon^2 \|X - H\|_2^2$.

The above follows from the fact that $|\textsc{Poss}(H) \setminus \mathcal{V}| \leq 2B \log n$. Consider $\mathcal{V}_0$ and $\mathcal{W}(X - H)$; since H can change at most $2B \log n$ coefficients, $\mathcal{W}(X - H)$ has at least $2B\epsilon^{-2} \log n$ coefficients of $\mathcal{V}_0$, each of which is larger than any of the $2B \log n$ coefficient in $\textsc{Proj}(X, \textsc{Poss}(H) \setminus \mathcal{V})$ (since the\operation took out the potentially large coefficients). Therefore, $\|\mathcal{W}(X - H)\|_2^2 \leq \epsilon^{-2} \|\textsc{Proj}(X, \textsc{Poss}(H) \setminus \mathcal{V})\|_2^2$. Now by Parseval, $\|\mathcal{W}(Y')\|_2 = \|Y'\|_2$ for any Y', and thus the lemma follows.

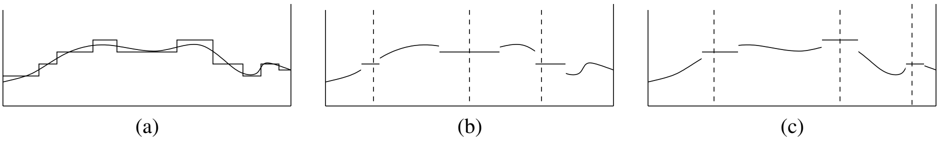


FIGURE 84.4 The use of the flattened function.

Definition 84.6

Given a B-bucket histogram H, and a set of hierarchically closed set of basis vectors $\mathcal{V}$, define $\text{FLAT}(X, \mathcal{V}, H) = X - \text{PROJ}(X, \text{POSS}(H)\backslash\mathcal{V})$*, i.e., all the places where H differs from the projection defined on $\mathcal{V}$ are* **flattened**.

It is immediate if $Y = X - \text{FLAT}(X, \mathcal{V}, H)$, then from Lemma 84.3 $\|Y\|_2 \leq \epsilon\|X - H\|_2$. Now, from the triangle inequality we have $\|X - H\|_2 + \|Y\|_2 \geq \|\text{FLAT}(X, \mathcal{V}, H) - H\|_2 \geq \|X - H\|_2 - \|Y\|_2$. Summarizing,

Lemma 84.4

If H_f minimizes $\|\text{FLAT}(X, \mathcal{V}, H) - H\|_2$ then $\|X - H_f\|_2$ is a $\frac{1+\epsilon}{1-\epsilon}$ approximation of the optimum error $\|X - H^*\|_2$.

We first show that if we "flatten" the signal at places where the coefficients are not large, then the error of a histogram in estimating the original signal X remains approximately the same if we use the same histogram to estimate the "flattened" signal.

Finding $\min_H \|\textbf{FLAT}(X, \mathcal{V}, H) - H\|$ We would first find the large Haar wavelet coefficients corresponding to $\mathcal{V}_0$ in a streaming fashion. For each of the *four endpoints* (including the middle two values corresponding to the jump) of the wavelets, we store SUM, SQSUM. Then instead of using the function X in the optimization, we would use the function $\text{FLAT}(X, \mathcal{V}, H)$. The illustration is in Figure 84.4, part (a) shows the approximation by the large wavelet functions. Part (b) shows what exactly we are computing, given a set of boundaries for H. Note that we are not approximating by wavelets, we are merely using the flat part from the wavelets corresponding to where the boundaries of H lie.

By definition, $\text{FLAT}(X, \mathcal{V}, H) = X - \text{PROJ}(X, \text{POSS}(H)\backslash\mathcal{V})$, i.e., when we decide on a particular boundary u of H, $\text{FLAT}(X, \mathcal{V}, H)$ looks *flat* between the two boundary points v_1, v_2 defined by $\mathcal{V}$ between which the boundary u falls. Note that this means that for different H, H' the $\text{FLAT}(X, \mathcal{V}, H)$, $\text{FLAT}(X, \mathcal{V}, H')$ look different (c.f. Figure 84.4 [b] and [c]). Since $\text{FLAT}(X, \mathcal{V}, H)$ looks flat, let that flat height between v_1, v_2 be h, we know this h since we know $\text{PROJ}(X, \mathcal{V})$. Now, we also know that $\text{SUM}'[u] = \text{SUM}[v_1] + h(u - v_1)$, where SUM' refers to $\text{FLAT}(X, \mathcal{V}, H)$. Likewise, $\text{SQSUM}'[u] = \text{SQSUM}[v_1] + h^2(u - v_1)$. At this point, we have all the pieces of the algorithm, we run the offline algorithm mentioned in Section 84.3. As the algorithm is described, our space requirement appears to be $O(n)$, but we can avoid that and use $O(B\epsilon^{-2}\log n)$ space corresponding to the size of $\text{PROJ}(X, \mathcal{V})$. We omit the details in the interest of space. The running time is $O(n + B^3(\log n + \frac{1}{\epsilon^2})\log^2 n)$, the extra $\log n$ appears from the fact that given u, we need to find v_1, v_2, h which takes $O(\log n)$ time. This immediately allows us to conclude that:

Theorem 84.10

We can compute a $1 + \epsilon$ approximation for the optimal B-bucket histogram under ℓ_2^2 error in a single pass in time $O(n + B^3(\log n + \frac{1}{\epsilon^2})\log^2 n)$ and space $O(B\epsilon^{-2}\log n)$.

Notes

Equiwidth histograms were introduced by Kooi [11]. Muralikrishna and DeWitt [12] considered equidepth/equiheight histograms, which are essentially quantiles. The piecewise constant definition arises from the work of Ioannidis and Christodoulakis [13] and led to the work of Ioannidis [14]. Ioannidis and Poosala [15] proposed several different measures and the ℓ_2^2/V-Optimal measure was introduced by Poosala et al. [16]. For a history of histograms, see Ref. [17].

The $O(n^2B)$ time $O(nB)$ space dynamic programming algorithm was given by Jagadish et al. [18]. They also showed that a $(B+\ell)$-bucket histogram, which has the same error as the optimal B-bucket histogram, can be constructed in $O(n^2B/\ell)$ time. The running time remained quadratic even if the approximation algorithm was allowed a constant factor larger resources. Guha et al. [19] gave the first FPTAS running in $O(\frac{nB^2}{\epsilon}\log n)$ time while preserving the number of buckets. Guha and Koudas [20] considered the question of constructing histograms over sliding window data streams and showed that a data structure can be maintained in $O(1)$ time, such that a $(1+\epsilon)$ approximation of the optimal histogram of the last n values can be constructed on demand in $O(\frac{B^3}{\epsilon^2}\log^3 n)$ time. The net result is an $O(n + poly(B, \epsilon, \log n)$ algorithm for histogram construction. Gilbert et al. [21] gave a polytime approximation scheme for the stronger dynamic/update model. The result holds for all ℓ_p norms with $0 < p \le 2$. Guha et al. [22] gave the first $\tilde{O}(B)$ space histogram algorithm for the weaker time series streaming model. This is interesting since B is not always a "small" constant and space is the premium in a streaming model; however, this algorithm only works for the ℓ_2 error. Guha et al. [23] considered the relative error and introduced the block-by-block algorithm and the amortized analysis. The journal version of Ref. [19], in Ref. [24], subsumed and improved several of these algorithms and provided the experimental validation of the approximation algorithms. Guha [25] improved Ref. [18] and gave an $O(n^2B)$ time $O(n)$ space algorithm as well as improved the space complexity of Refs. [22,24]. The result concerning the ℓ_∞ histograms in Section 84.3.3 are from Ref. [26]. The discussion on piecewise polynomials and other extensions in Section 84.3.2 is from Refs. [19,24]. Donjerkovic et al. [27] considered the χ^2 measure.

The above discussion applied to point queries. Range queries form an important class of queries and pose significantly more complicated optimizations. Several early papers consider only the restricted version where we store the mean of the values in a bucket, which is suboptimal. Koudas et al. [28] considered hierarchical queries and gave an algorithm that runs in time $O(|T|n^6B^2)$ for a hierarchy of size $|T|$. Gilbert et al. [29] gave a pseudo-polynomial time algorithm for the case when all $\binom{n}{2}$ ranges are present. Guha et al. [30] gave a sparse set system–based algorithm that improved the running time of the hierarchical case to $O(n + |T|B^2n^\gamma)$ but used $12B/\gamma$ buckets, while preserving the optimum guarantee with respect to B buckets. However, all these algorithms considered the restricted model where we are restricted to store the mean of a set of values as a representative. Muthukrishnan and Strauss [31] considered the unrestricted version when all $\binom{n}{2}$ ranges are present and gave an $O(n^2B)$ algorithm that uses $2B$ buckets and guarantees less or equal error compared to the best B-bucket histogram. They also gave an $O(B^3N^4/\epsilon^2)$ time approximation scheme for this case. The space requirement of most of these algorithms were improved in [25].

Wavelets have a rich history dating back to the work of Haar (1910). However, they became significantly more popular in early 1990s primarily due to the application in image analysis [32,7], and the seminal work of Daubechies [8]. In the context of database systems, wavelets were introduced by Matias et al. [33]. Their paper proposed greedy algorithms for optimum wavelet selection for several error measures including ℓ_1. Chakraborty et al. [34] consider using wavelets for modeling time series data. Gilbert et al. [35] gave the streaming algorithm for the ℓ_2 case for Haar wavelets. Gilbert et al. [21] extend the ℓ_2 result to the stronger model of update streams. Garofalakis and Gibbons [36] considered constructing synopses based on randomized rounding techniques. Garofalakis and Kumar [37] considered the restricted version (of retaining the coefficients) for ℓ_∞ (and similar measures) and gave an $O(n^2B)$ time and space algorithm. This was improved to $O(n^2)$ time and $O(n)$ space in Ref. [25], for a broad range of error measures, including workloads. All the above are in the context of Haar wavelets.

Matias and Urieli [38] considered the problem of designing a basis for weighted ℓ_2 measures such that greedy coefficient selection was optimum for that basis. Guha and Harb [39] gave the first-approximation schemes for the original (unrestricted) optimization problem, which is discussed in Section 84.4.3. For ℓ_p error measures with $p > 2$ (e.g., ℓ_∞), the algorithm can be implemented as a small space streaming algorithm for Haar wavelets. They also proved the upper bound on the gap between the optimum of the restricted version and the unrestricted optimum, which is discussed in Section 84.4.4. In case of range queries using wavelets, Matias and Urieli [40] show that a scaled greedy strategy is provably optimal when all $\binom{n}{2}$ ranges are equally likely. Gilbert et al. [29] also discusses a similar scenario. Guha et al. [41] consider the hierarchical case in the restricted model.

The algorithm in Section 84.5 is based primarily on the ideas in Ref. [22]. The improved space bounds follows from Ref. [25].

References

[1] Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., and Price, T. G., Access path selection in a relational database management system, *Proc. ACM SIGMOD*, 1979, p. 23.

[2] Aboulnaga, A. and Chaudhuri, S., Self tuning histograms: Building histograms without looking at data, *Proc. ACM SIGMOD*, 1999, p. 181.

[3] Bruno, N., Gravano, L., and Chaudhuri, S., STHoles: a workload aware multidimensional histogram, *Proc. ACM SIGMOD*, 2001, pp. 211–222.

[4] Datar, M., Gionis, A., Indyk, P., and Motwani, R., Maintaining stream statistics over sliding windows, *Proc. SODA*, 2002, p. 635.

[5] Manku, G. S., Rajagopalan, S., and Lindsay, B., Approximate medians and other quantiles in one pass and with limited memory, *Proc. ACM SIGMOD*, 1998, p. 426.

[6] Greenwald, M. and Khanna, S., Space-efficient online computation of quantile summaries, *Proc. of ACM SIGMOD*, 2001, pp. 58–66.

[7] Mallat, S., *A Wavelet Tour of Signal Processing*, Academic Press, New York, 1999.

[8] Daubechies, I., *Ten Lectures on Wavelets*, SIAM, Philadelphia, 1992.

[9] Bentley, J. L., Multidimensional divide-and-conquer, *CACM*, 23(4), 214, 1980.

[10] Guha, S., Mishra, N., Motwani, R., and O'Callaghan, L., Clustering data streams, *Proc. FOCS*, 2000, 359.

[11] Kooi, R., The optimization of queries in relational databases, *Case Western Reserve University*, 1980.

[12] Muralikrishna, M. and DeWitt, D. J., Equi-depth histograms for estimating selectivity factors for multidimensional queries, *Proc. ACM SIGMOD,* 1988, p. 28.

[13] Ioannidis, Y. and Christodoulakis, S., Optimal histograms for limiting worst-case error propagation in the size of join results, *ACM Trans. Database Syst.*, 18(4), 709, 1993.

[14] Ioannidis, Y. E., Universality of serial histograms, *Proc. VLDB Conf.*, 1993, p. 256.

[15] Ioannidis, Y. and Poosala, V., Balancing histogram optimality and practicality for query result size estimation, *Proc. ACM SIGMOD*, 1995, p. 233.

[16] Poosala, V., Ioannidis, Y., Haas, P., and Shekita, E., Improved histograms for selectivity estimation of range predicates, *Proc. ACM SIGMOD*, 1996, p. 294.

[17] Ioannidis, Y. E., The history of histograms (abridged), *Proc. VLDB Conf.*, 2003, p. 19.

[18] Jagadish, H. V., Koudas, N., Muthukrishnan, S, Poosala, V., Sevcik, K. C., and Suel, T., Optimal histograms with quality guarantees, *Proc. VLDB Conf.*, 1998, p. 275.

[19] Guha, S., Koudas, N., and Shim, K., Data streams and histograms, *Proc. STOC*, 2001, p. 471.

[20] Guha S. and Koudas, N., Approximating a data stream for querying and estimation: Algorithms and performance evaluation, *Proc. ICDE*, 2002, p. 567.

[21] Gilbert, A. C., Guha, S., Indyk, P., Kotidis, Y., Muthukrishnan, S., and Strauss, M., Fast, small-space algorithms for approximate histogram maintenance, *Proc. ACM STOC*, 2002, p. 389.

[22] Guha, S., Indyk, P., Muthukrishnan, S., and Strauss, M., Histogramming data streams with fast per-item processing, *Proc. ICALP*, 2002, p. 681.

[23] Guha, S., Shim, K., and Woo, J., REHIST: Relative error histogram construction algorithms, *Proc. VLDB Conf.*, 2004, p. 300.

[24] Guha, S., Koudas, N., and Shim, K., Approximation and streaming algorithms for histogram construction problems, *ACM TODS,* 31(1), 396–438, 2006.

[25] Guha, S., Space efficiency in synopsis construction problems, *Proc. VLDB Conf.*, 2005, p. 409.

[26] Guha, S. and Shim, K., ℓ_∞ histograms, Technical Report, 2005.

[27] Donjerkovic, D., Ioannidis, Y. E., and Ramakrishnan, R., Dynamic histograms: capturing evolving data sets, CS-TR 99-1396, University of Wisconsin, Madison, WI, 1999.

[28] Koudas, N., Muthukrishnan, S., and Srivastava, D., Optimal histograms for hierarchical range queries, *Proc. ACM PODS*, 2000, p. 196.

[29] Gilbert, A. C., Kotidis, Y., Muthukrishnan, S., and Strauss, M., Optimal and approximate computation of summary statistics for range aggregates, *Proc. ACM PODS*, 2001, pp. 227–236.

[30] Guha, S., Koudas, N., and Srivastava, D., Fast algorithms for hierarchical range histogram construction, *Proc. ACM PODS*, 2002, p. 180.

[31] Muthukrishnan, S. and Strauss, M., Rangesum histograms, *Proc. SODA*, 2003, p. 233.

[32] Jacobs, C. E., Finkelstein, A., and Salesin, D. H., Fast multiresolution image querying, *Proc. Computer Graphics Conf.*, 1995, p. 277.

[33] Matias, Y., Vitter, J. S., and Wang, M., Wavelet-based histograms for selectivity estimation, *Proc. ACM SIGMOD*, 1998, p. 448.

[34] Chakrabarti, K., Garofalakis, M. N., Rastogi, R., and Shim, K., Approximate query processing using wavelets, *Proc. VLDB Conf.*, 2000, p. 111.

[35] Gilbert, A., Kotadis, Y., Muthukrishnan, S., and Strauss, M., Surfing wavelets on streams: one pass summaries for approximate aggregate queries, *Proc. VLDB Conf.*, 2001, p. 79.

[36] Garofalakis, M. N. and Gibbons, P. B., Probabilistic wavelet synopses, *ACM TODS*, 29, 43, 2004.

[37] Garofalakis, M. and Kumar, A., Deterministic wavelet thresholding for maximum error metric, *Proc. PODS*, 2004, p. 166.

[38] Matias, Y. and Urieli, D., Optimal workload-based weighted wavelet synopses, *Proc. ICDT*, 2005, p. 368.

[39] Guha S. and Harb, B., Approximation algorithms for wavelet transform coding of data streams, *Proc. SODA*, p. 2006.

[40] Matias, Y. and Urieli, D., On the Optimality of the Greedy Heuristic in Wavelet synopses for range queries, Technical Report, Tel Aviv University, 2005.

[41] Guha, S., Park, H., and Shim, K., Wavelet synopsis for hierarchical range queries with workloads, 2005 (submitted).

85

Digital Reputation for Virtual Communities

Roberto Battiti
University of Trento

Anurag Garg
University of Trento

85.1 Introduction

A virtual community can be defined as a group of people sharing a common interest or goal who interact over a virtual medium, most commonly the Internet. Virtual communities are characterized by an absence of face-to-face interaction between participants which makes the task of measuring the trustworthiness of other participants harder than in nonvirtual communities. This is due to the anonymity that the Internet provides, coupled with the loss of audiovisual cues that help in the establishment of trust. As a result, digital reputation management systems are an invaluable tool for measuring trust in virtual communities.

Trust is an important component of all human interactions whether they take place online or not. There is an implicit assumption of trust in each interaction that we participate in that involves some investment on our part. The Merriam-Webster dictionary defines trust as *assured reliance on the character, ability, strength, or truth of someone or something*. Even more pertinent to virtual communities is an alternative definition which states that trust is a *dependence on something future or contingent; reliance on future payment for property (as merchandise) delivered*. These definitions illustrate that the basis of trust is an expectation of future payment or reward and that the transaction partner will behave in an honest fashion and fulfill their obligations.

The processes behind the creation of trust can be classified into four broad categories:

1. *Personality-based trust.* A person's innate disposition to trust others.
2. *Cognitive trust.* Through recognition of the characteristics of the transaction partner.
3. *Institution-based trust.* A buyer's perception that effective (third-party) institutional mechanisms are in place to facilitate transaction success [1].
4. *Transaction-based trust.* That relies on a participant's past behavior to assess their trustworthiness.

Personality-based trust does not have a significant use in virtual communities where decisions on trust are made algorithmically. Most cognitive factors that form trust in real-life interactions such as the manner of a person, their body language, and the intonations of their speech are also absent in virtual communities. However, there are alternative forms of cognition that can be used instead. These are almost invariably based on the virtual identity of the community member. In most *purely* online contexts—as opposed to contexts where the online identity is linked to a non-online identity such as with online banking—there are virtually no costs to creating a new virtual identity. Obtaining an e-mail address on any of the free web-based e-mail services such as Yahoo, Hotmail, Gmail is trivial and with each one comes a new virtual identity. While these services are increasingly adopting strategies such as requiring text perception to counter automated registration programs, it is not difficult to create, say, a dozen accounts in 10 min. And as the e-Bay scam case [2] showed, that is all a malicious user needs to surmount simple feedback systems and commit online fraud.

Enforcing trust in virtual communities is hard not only because of the difficulties in recognizing the trustworthiness of participants but also because of the lack of adequate monitoring and appropriate sanctions for dishonest behavior. This lack of *institution-based trust* is because there are not enough trusted third parties with the power to punish to ensure honesty of all the players [3]. This problem is exacerbated in decentralized virtual communities and electronic marketplaces. An organization like e-Bay with a centralized infrastructure can act as the trusted third party to at least store feedback information in a reliable fashion even though the information itself may not be reliable. The problem becomes much harder with decentralized systems such as peer-to-peer (P2P) networks where the absence of a centralized authority is a defining feature of the system.

Hence, it appears that the best strategy for creating trust in virtual communities is through *transaction-based trust* where feedback on participants' past behavior is collected and aggregated to decide their trustworthiness. A transaction-based trust strategy is far from perfect and suffers from many of the same shortcomings as the other strategies. For instance, the lack of verifiable identities [4] can make a transaction-based system vulnerable to manipulation. However, as many recent proposals have shown [5–10] such strategies are not contingent upon a trusted third party and the community as a whole provides the institutional basis for trust. Hence, if the problem of identity can be solved, transaction-based systems are capable of providing the solution for virtual communities.

Transaction-based trust creation strategies are also commonly known as reputation-based trust management systems or just *reputation systems*. Some authors use the term reputation systems narrowly to include only those systems that monitor past transactions of participants to compute their trustworthiness. This information is then shared with other participants who decide whether or not to interact with the target participant on its basis. We use the term in a more general sense to include recommendations systems that recommend items as well as systems that perform distributed authentication by inferring trustworthiness. All these systems have one feature in common: the collection and aggregation of feedback in order to rate objects or people.

Reputation systems research lies at the intersection of several disciplines including evolutionary biology [11–13], economics (game theory and mechanism design) [14–17], sociology (social network theory [18–20]), and computer science (e-commerce, P2P systems, cryptography, etc.). In this chapter, we survey the approaches taken by researchers from these disciplines to provide the context for digital reputation schemes. We begin by listing the various applications of digital reputation systems. This is followed by a discussion of what motivates the participants of a virtual community to cooperate with each other. We then look at what are the requirements of a good reputation system. This is followed by a more detailed analysis of the components of reputation systems. We classify different types of feedback and their role in constructing reputations. We discuss second-order reputation and the motivations for providing feedback. Reputation modifying factors such as transaction context are looked at next followed by a discussion on how reputations can be interpreted. We conclude by looking at several specific digital reputation systems that have been proposed.

85.2 Applications of Reputation Management

The main uses of digital reputation management systems in virtual communities are:

1. Incentivizing cooperation
2. Identifying and excluding malicious entities

3. Authenticating users in a distributed manner
4. Providing recommendations

Resnick et al. [21] define three requirements for a reputation system: (1) to help people decide whom to trust, (2) to encourage trustworthy behavior and appropriate effort, and (3) to deter participation by those who are unskilled or dishonest. To this we can add the requirements that a reputation system (4) must preserve anonymity associating a peer's reputation with an opaque identifier, and (5) have minimal overhead in terms of computation, storage, and infrastructure.

The participation of an individual in a virtual community strongly depends on whether and how much benefit they expect to derive from their participation. If an individual feels that they will not gain anything from joining the community, they are unlikely to participate. Hence, a good distributed system must be designed such that it incentivizes cooperation by making it profitable for users to participate and withhold services from users that do not contribute their resources to the system.

An equally serious challenge to distributed systems and virtual communities comes from users who act in a malicious fashion with the intention of disrupting the system. Examples of such malicious behavior include users who pollute a file-sharing system with mislabeled or corrupted content, nodes that disrupt a P2P routing system to take control of the network, nodes of a Mobile Ad hoc NETwork (MANET) that misroute packets from other nodes [10] and even spammers [22] who are maliciously attacking the e-mail community. Therefore, a central objective of all digital reputation schemes is to identify such users and punish them or exclude them from the community. This exclusion is usually achieved by allowing members to distinguish between trustworthy and untrustworthy members. An untrustworthy member will not be chosen for future interactions. In contrast with incentivizing cooperation that motivates truth from participants, this is based on punishing falsehoods.

Another use for digital reputation systems is for distributed authentication. Distributed authentication does not rely on strict hierarchies of trust such as those using certification authorities and a centralized public key infrastructure that underpin conventional authentication. Such networks capture trust relationships between entities. Trust is then propagated in the network so that the trust relationship between any two entities in the network can be inferred. PGP [23], GnuPGP, and OpenPGP-compatible [24] systems all use webs of trust for authentication. Closely related are virtual social networks including the Friend of a Friend (FOAF) system, LinkedIn, Tribe.net, Orkut, and Friendster that use some or the other form of trust propagation.

Recommendation systems are another application that relies on principles similar to digital reputation systems. Instead of computing the trustworthiness of a participant, recommendation systems compute the recommendation score of objects based on collective feedback from other users. These systems are in widespread commercial use and examples include the Amazon recommendation system and the IMDB movie recommendations. In recommendation systems, objects instead of members of a virtual community are rated. The members may then be rated on the quality of feedback they provide just like in reputation systems.

When the number of objects to be rated is usually large compared to the user-base, the collected data can be very sparse. Collaborative-filtering-based recommendation systems use the collective feedback to compute the similarity in the ratings made by any two users and using this similarity to weigh the rating of one user when predicting the corresponding choice for the other user. There are many ways in which this similarity can be computed. One method uses the Pearson's correlation coefficient [25]:

$$w_{u,i} = \frac{\sum_j (v_{u,j} - \bar{v}_u)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{u,j} - \bar{v}_u)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}} \tag{85.1}$$

where $w_{u,i}$ computes the similarity between users u and i on the basis of their votes for all objects j that both have voted for and $\bar{v}_u$ and $\bar{v}_i$ denote, respectively, the average ratings given by users u and i.

Another method is to use vector similarity [26]:

$$w_{u,i} = \sum_{j} \frac{v_{u,j}}{\sqrt{\sum_{k \in I_u} v_{u,k}^2}} \frac{v_{i,j}}{\sqrt{\sum_{k \in I_i} v_{i,k}^2}} \tag{85.2}$$

where $v_{u,j}$ is the rating given by user u for object j as before and I_u the set of objects for which user u has a rating. Other model-based methods exist that use Bayesian network models or clustering models to group users together and use group membership to predict what a user may vote for a given object. Clustering models have also been shown to be useful in reputation systems for eliminating spurious ratings [27] made by malicious users in an electronic marketplace.

85.3 Motivating Cooperation

Motivating cooperation among participants has been a subject of research since long before the first virtual communities were formed. As long as there are shared resources, there will be users who are tempted to use more than their fair share for their own benefit even if it is at the expense of the community as a whole. This conflict between individual interests and the common good is exemplified in the “tragedy of the commons,” a term coined by Hardin [28], who used the overgrazing of the English “commons” (property that was shared by the peasants of a village) as an example. In reputation systems research such selfish users are termed as *free-riders* or *free-loaders.*

An example of free-riding can be found in MANETs. MANETs function on the premise that nodes will forward packets for each other even though forwarding a packet consumes power. However, if there are free-riders that inject their own packets in the network but do not forward any packets from other nodes, they can exploit the cooperative nature of other users.

Another example of free-riding in virtual communities is found in early file-sharing systems, where an individual is allowed to download files without being required to contribute any files for uploading. In their measurement study of Napster and Gnutella, Saroiu et al. [29] reported that this resulted in significant fractions of the population indulging in client-like behavior. They reported that a vast majority of modem users were free-riders who did not share any files and only downloaded files offered by others. However, equally interesting is the fact that many users with high bandwidth connections indulged in server-like behavior and offered their own files for sharing but did not download any files at all. This unselfish behavior is the flip side of free-riding where users in a virtual community indulge in *altruism.*

Altruism has been studied by evolutionary biologists [11–13] in humans and other primates. These authors have sought to explain altruistic behavior by arguing that it signals “evolutionary fitness.” By behaving generously an individual signals that it has “plenty to spare” and is thus a good mating choice. In the context of virtual communities, altruistic behavior is motivated in part by a desire to signal that interacting with the individual is likely to be beneficial in other contexts as well.

Another strand of research comes from economists who have studied cooperation by setting up non-zero sum games and determining the equilibria that result both through analytic game theory and through simulation. In game theory, it is usually assumed that players are “rational” or “selfish,” that is, they are only interested in maximizing the benefit they obtain with no regard to the overall welfare of the community. This is distinct from an “irrational” player whose utility function depends on something more than just their own benefit and whose behavior cannot be predicted. An example of irrational players are “malicious” players who actively wish to harm other players or the game as a whole even if it means reducing their own personal benefit.

The “game” that is typically used for modeling the problem of cooperation is the Prisoner’s Dilemma [30,31]. The classic prisoner’s dilemma concerns two suspects who are questioned separately by the police and given a chance to testify against the other. If only one prisoner betrays the other, the betrayer goes free while the loyal prisoner gets a long sentence. If both betray each other they get a medium sentence and if both stay silent they get a small sentence. Hence, if both prisoner’s are “selfish” and have no regard for the other prisoner, we see the best strategy for a prisoner is always to betray the second prisoner, regardless of what the other prisoner chooses. If the second prisoner confesses, the first prisoner must confess too otherwise the first prisoner will get a long sentence. And if the second prisoner does not confess, the first prisoner can get off free by confessing as opposed to getting a short sentence by not confessing.

If the game is played only once, the solution is obvious. “Always Defect” is the dominant strategy.[1] Axelrod [14] studied an interesting extension to the classic problem which he called the iterated prisoner’s dilemma. Here, members of a community play against each other repeatedly and retain memory of their past interactions. Hence, they have an opportunity to punish players for their past defections. Axelrod set up an experiment with various strategies submitted by fellow researchers playing against each other. He discovered that “greedy” strategies tended to do very poorly in the long run and were outperformed by more “altruistic” strategies, as judged by pure self-interest. This was true as long as cheating was not tolerated indefinitely. By analyzing the top-scoring strategies, Axelrod stated several conditions necessary for a strategy to be successful. A successful strategy should be (1) *Nice*: it will not defect before an opponent does. (2) *Retaliating*: it will always retaliate when cheated. A blind optimist strategy like “Always Cooperate” does not do well. (3) *Forgiving*: in order to stop endless cycles of revenge and counterrevenge a strategy will start cooperating if an opponent stops cheating. (4) *Non-envious*: a strategy will not try to outscore an opponent. Axelrod found that the best deterministic strategy was “tit-for-tat” in which a player behaved with its partner in the same way as the partner had behaved in the previous round.

The prisoner’s dilemma is a game with a specific reward function that encourages altruistic behavior and discourages selfish behavior. Let T be the temptation to defect, R the reward for mutual cooperation, P the Punishment for mutual defection, and S the Sucker’s punishment for cooperating while the other defects. Then, the following inequality must hold:

$$T > R > P > S \tag{85.3}$$

In the iterated game, yet another inequality must hold:

$$T + S < 2R \tag{85.4}$$

If this is not the case, then two players will gain more in the long run if one cooperates and the other defects alternately rather than when both cooperate. Hence, there will be no incentive for mutual cooperation. Recall that the objective of a rational player is to maximize their individual score and not score more than the opponent.

Other virtual communities may have different cost and reward functions. The game-theoretic approach is to devise strategies to maximize individual utility in a fixed game. However, if a community is designed so that its interests as a whole are not aligned with those of an individual, the system will be “gamed” by the rational users leading to the ultimate failure of the virtual community. Mechanism design [15] deals with the design of systems such that players’ selfish behavior results in the desired system-wide goals. This is achieved by constructing cost and reward functions that encourage “correct” behavior.

Another example of incentivizing cooperation can be found in collaborative content distribution mechanisms [32–34] such as BitTorrent [35], where peers cooperate to download a file from a server by downloading different chunks in parallel and then exchanging them with each other to reconstruct the entire file. In fact, BitTorrent implements a “tit-for-tat” strategy to prevent free-riding. Basic reputation schemes have also been implemented in second generation file-sharing systems such as Kazaa that measure the participation level of a peer based on the number of files the peer has uploaded and downloaded. Peers with a higher participation level are given preference when downloading a file from the same source.

85.4 Design Requirements for a Reputation System

While mechanism design may help achieve overall system goals through proper incentivization, it is ineffective against deliberately malicious (or irrational) participants. To solve this problem a number of reputation systems have emerged. These operate by allowing a user to rate the transactions they have had

[1] A dominant strategy always give better payoff than another strategy regardless of what other players are doing.

with other users. This feedback is collected and aggregated to form a reputation value which denotes the trustworthiness of a user. This reputation information is made available to requesting users who then make their decisions on whether or not to interact with a given user based on its reputation. Several competing schemes for aggregating this feedback have been proposed [5–9,36,37].

The architectural and implementation details of the aggregation mechanism depend on the underlying network on which the virtual community is based. When the community is built on top of a traditional client-server network, a trusted third party exists which can be relied on to collect and aggregate opinions to form a global view. e-Bay feedback, Amazon customer review and the Slashdot-distributed moderation systems are all examples where feedback from users is stored in a centralized trust database. The aggregation is performed in this centralized database and all users have access to the global reputations thus computed. When the community is built on top of a P2P network, the challenges of managing feedback become much harder. There is no centralized, trusted, reliable, always-on database and the collection, storage, aggregation, and dispersal of trust information must be done in a distributed way. Relying on third parties for storage and dissemination also makes the system vulnerable to tampering and falsification of trust information in storage and transit. Moreover, the system must also provide redundancy because users may drop out of the network at any time.

Hence, there are two separate but interrelated challenges that must be overcome by any distributed trust management system. The first is the choice of an appropriate trust metric that accurately reflects the trustworthiness of users and is resistant to tampering and other attacks. This was recognized by Aberer and Despotovic as "the semantic question: which is the model that allows us to assess trust [of an agent based on that agents behavior and the global behavior standards]" [5]. The second is designing a system architecture that is robust against the challenges mentioned above or the "data management" problem.

At this point, it is useful to ask why a reputation system would work in a virtual community and what may cause it to fail. The reasons for potential success are: (1) The costs of providing and distributing reputations are negligible or zero. (2) The infrastructure for decentralized aggregation and dissemination already exists in the form DHT-based[2] routing systems [38,39]. (3) It is easy to build redundancy in the reputation system. And the potential failings are: (1) Users may lie about the feedback they provide. (2) Users may not bother to give feedback. (3) Untrustworthy users may mask their behavior or retaliate against negative feedback by sending negative feedback about other users. (4) Users may try to game the system by building up their reputation by acting honestly over several small transactions followed by cheating in a large transaction in a process known as *milking.* (5) Users may form malicious groups that give false positive ratings to each other to boost each other's reputations. (6) Users may re-enter the system with a new identity to conceal their past behavior. Hence, a good reputation system must try to overcome these potential failings.

85.5 Analysis of Reputation System Components

85.5.1 Direct versus Indirect Evidence

Two distinct types of evidence are usually combined by reputation systems to form the reputation of a user. These are (1) *direct evidence*, which consists of a user's first-hand experiences of another user's behavior and (2) *indirect evidence* which is the second-hand information that is received from other nodes about their own experiences. If only first-hand information were used to decide the reputation of another peer, the sparseness of feedback would be a problem because in large virtual communities the interaction matrix is usually very sparse. A user would not be able to make a trust judgment on another user with whom they have never interacted before. Hence, users must rely on indirect evidence to compute the reputation of users that are new to them.

[2] In systems using distributed hash tables or DHTs, objects are associated with a key (usually produced by hashing an object ID such as the filename) and each node in the system is responsible for all objects associated with a key that falls in a certain range.

In a centralized system such as e-Bay all indirect evidence is collected at the central trust database and is made available to other users. In a decentralized system indirect evidence can be shared in a number of ways. The interested user may ask for indirect evidence from its neighbors or other users it trusts [6]. The indirect evidence may also be propagated to all users in the network using a recursive mechanism [7]. *Designated Agents*[3] may also be chosen from within the community to store indirect evidence which can then be furnished to requesting users [5,8,9]. In the latter schemes, designated agents responsible for specific users in the system are chosen using distributed hash tables. Multiple agents are chosen to ensure redundancy in case an agent leaves the network or tries to falsify the stored evidence.

85.5.2 Second-Order Reputation

Using second-hand information in a decentralized system leads to another problem. How can a user know that the provider of the second-hand information is telling the truth? We can think of an individual's reputation for providing accurate reputation information on others as their second-order reputation information. Second-order reputation is often termed as *credibility* as it measures the truthfulness of an individual as a provider of information.

The problem was first addressed by Aberer and Despotovic [5] who were among the first to use a decentralized reputation storage system. They use the trustworthiness of an agent (first-order reputation), to decide whether to take its feedback into account. The feedback from agents who are deemed trustworthy is included and that from untrustworthy agents is excluded. Kamvar et al. [7] use the same strategy of using first-order reputation as second-order reputation as well. However, while it is reasonable to assume that an individual who cheats in the main market cannot be relied upon for accurate feedback, the reverse is not necessarily true. An individual can act honestly in the main market and enjoy a high reputation and at the same time provide false feedback to lower the reputation of others in the community who may be his/her competitors. An example where such a strategy would be advantageous is a hotel rating system where a hotel may provide very good service and thus have a high reputation but at the same time may give false feedback about its competitors to lower their reputation. Hence its credibility is very different from its reputation.

The solution is to recognize credibility as different from first-order reputation and compute it separately. Credibility can be computed in several different ways. Credibility values can be expressed explicitly and solicited separately from reputation values. However this leads to a problem of endless recursion. To ensure that users do not lie about other users credibility, we need third-order reputation to measure an individual's reputation for providing accurate second-order reputation information and so on.

Credibility can also be computed implicitly. This can be done in two ways. In the first method, inspired by collaborative filtering, the similarity of user j's opinions to those of user i are used to compute the credibility of user j in the eyes of user i (C_{ij}). This can be done using Pearson's correlation coefficients as used by Resnick et al. [25] or by using vector similarity like in Breese et al. [26]. PeerTrust [8] uses yet another similarity metric that resembles the Pearson coefficient. In this method, if user i's opinions are often at variance with those of user j, i will have a lower credibility value for j and vice versa.

The second approach for implicit credibility computation measures the credibility of a user by taking into account the agreement between the feedback furnished by the user and the views of the community as a whole. The community average view can be represented by the reputation value computed from the feedback from all the reporting users. If a user gives wrong feedback about other users, that is, his or her feedback is very different from the eventual reputation value computed, its credibility rating is decreased and its subsequent reports have a reduced impact on the reputation of a user. This method of computing credibility has an advantage in that it is more scalable and it can operate in decentralized systems where a complete record of all opinions expressed by other individuals may not be available due to privacy

[3] The term *Designated Agents* was coined by the authors and includes all systems where one or more users are made responsible for storing and sharing another user's reputation information by the system.

concerns. However, there is a danger of "group-think" in such systems. There is strong encouragement for an individual to agree with the opinion of the group as a whole as disagreements are punished by lowering the credibility of the individual who disagrees.

85.5.3 Motivation for Providing Feedback

A closely related issue is how to motivate community members to provide feedback to others when providing feedback consumes their own resources. In some respects this problem is the same as that of motivating cooperation between community members as discussed above. However, there are some important differences. In a designated agent system, designated agents must use their own resources to store, compute, and report reputation values for other members. Most literature assumes that agents will perform this task because they are "altruistic" or because they too will benefit from a designated agent system. However, in a large network, there is little incentive for a particular individual to expend resources to maintain the reputation system. How do we prevent such an individual from free-riding the reputation system itself? Moreover, from a game-theoretic perspective, not reporting feedback may be advantageous to an agent in a competitive situation. By reporting feedback to other agents, it is sharing information with them which if kept to itself may have given it an advantage.

One strategy to encourage truthful reputation reports is to create a side market for reputation where accurate reputation reports are rewarded with currency that can be traded for accurate reports from others [40]. Such a system needs to be structured in such a way that providing more feedback and providing honest feedback results in more credit. However this solution suffers from the recursion problem mentioned above as third-order market would need to be created and so on.

An alternative way to avoid the recursion problem is to incorporate the credibility of an individual in the reputation system as a separate variable as discussed in the previous section. If the credibility is computed through direct evidence only and is not shared with others [9] then the recursion problem can be avoided.

85.5.4 Motivation Is Not a Problem: Dissenting Views

A number of authors do not agree that motivation (of cooperation and of sharing feedback) is a problem at all. Fehr and Gächter [13] develop a theory of "altruistic punishment." They designed an experiment that excluded all explanations for cooperation other than that of altruistic punishment. They designed a McCabe style investment game [41] where players could punish their partners if they wished. However, punishing was costly both to the punisher (one point) and the punished (three points). Punishment induced cooperation from potential noncooperators thus increasing the benefit to the group as a whole even though it was costly to the punisher. For this reason punishment was an altruistic act. They concluded that negative emotions are the cause of altruistic punishment and that there is a tendency in humans to punish those that deviate from social norms.

Similarly, Gintis et al. [12] argue that behaving in an altruistic fashion sends a signal to other participants that interacting with that individual is likely to prove beneficial. This argument cuts at the heart of the game-theoretic notion of how rational agents operate. Rational agent behavior now becomes probabilistic in that an agent may act in an altruistic fashion in the hope of future reward instead of only interacting when an immediate benefit is expected. This interpretation also depends on whether the individuals in the system are humans or are automated agents that do not share the "altruistic" characteristics and behave in a strictly rational sense.

This has been used as evidence that altruism serves an agent's self-interest. It also explains why greedy strategy are outperformed by more altruistic strategies in Axelrod's experiment. Hence, there is some theoretical basis to the claim that a virtual community can be be self-correcting and will exclude bad participants.

85.5.5 Other Design Issues

A number of other design choices must be made in a reputation system.

Reputation context. There has been some research on whether reputation is contextual. It is often assumed that reputation is heavily dependent on context. This point of view was aptly expressed by Mui et al. [42]

> "Reputation is clearly a context-dependent quantity. For example, one's reputation as a computer scientist should have no influence on his or her reputation as cook."

However, in a contrary argument, Gintis et al. [12] suggest that compartmentalizing reputation too strictly can have a negative effect. Their contention is that altruistic behavior is motivated in part by a desire to signal that interacting with the individual in question will be beneficial in other contexts as well. They argue that the notion of reputation in a real world is far more fuzzy and incorporates generosity as well. A generous participant is more likely to be honest as well. This gives participants an incentive to behave in an altruistic fashion in addition to behaving in an honest fashion.

Transaction value. A reputation system must also be able to distinguish between small and large transactions. Buying a pencil online is not the same as buying a car. If all transactions are rated equally, an individual may exploit the system through a strategy called *milking* where they act honestly in a number of small transactions to build their reputation and then cheat in a large transaction without hurting their reputation too much. In PeerTrust [8] this problem is solved by incorporating a transaction context factor that weighs the feedback according to size, importance, and the recency of the transaction.

Interpreting reputation. Once the reputation of a user (or object in case of recommendation systems) has been computed, it can be used in several ways depending on the application context. In a file-sharing system [6] a peer may choose the peer with the highest reputation to download the file from. On Amazon, the recommendation system may prompt one to buy a book that one was not aware of before. GroupLens [25] helps decide which movie a user decides to watch.

Equally common are applications that demand a Boolean yes/no decision on "Should i trust j?". There are several methods by which this translation from an arbitrary range of reputation values to a binary value can be achieved. These include (1) using a deterministic threshold (I will trust you if your reputation value exceeds 6 on a scale of 10), (2) relative ranking (I will trust you if your reputation is in the top 10% of community members), (3) probabilistic thresholds (the probability that I trust you is a monotonically increasing function on the range of possible trust values), and (4) majority rounding (I will trust you if I trust a majority of people with reputation values close to your reputation).

Another approach [19,43] is to include information that allows a user to decide how much faith it should place in the reputation value. On e-Bay this information is the number of feedbacks a user has received. A user with a high reputation and a large number of feedbacks is much more trustworthy than one with a low number of feedbacks.

Benefits of high reputation. Many reputation systems particularly those proposed for file-sharing applications [5–7] do not consider the consequences of a peer having a high reputation. In file-sharing systems the most reputable peer in the network will be swamped with requests as all peers are going to want to download the resources from it. Hence peers with high reputations are "punished" for their high reputations by having to expend more of their resources to serve others instead of being benefited from their reputation. In this scenario the interests of the system are not aligned with that of individuals and the individual peers have no motivation to act honestly and thus increase their reputation.

To motivate individuals to try and acquire high reputations, there needs to be a mechanism by which nodes that have a high reputation are rewarded. In a file-sharing application this could be achieved through preferential access for higher reputation nodes to resources at other nodes.

Positive versus negative feedback. A reputation system may be based on either positive feedback only, negative feedback only or a combination of both. The disadvantage of a negative feedback only system [5] is that each new entrant into the system has the highest possible reputation. Hence, a misbehaving individual may create a new identity and shed their bad reputations to start afresh. Using only positive feedback, in contrast, makes it hard to distinguish between a new user and a dishonest user. If old users choose not to interact with any users without a minimum level of positive feedback, a new user may thus find itself frozen out of the group and interacting only with malicious users.

Identities. A reputation system that allows unlimited creation of new identities is vulnerable to manipulation [4]. Hence, there must be a cost associated with creating an identity. However, at the same time this cost must not be so large as to discourage newcomers from joining. Friedman and Resnick argue that in social situations there is inevitably some form of initiation dues [44]. They also find that while these dues are inefficient, especially when there are many newcomers to a community, no other strategy does substantially better in terms of group utility.

85.6 Some Reputation Systems

We now look at some reputation management algorithms that have been proposed in recent years.

85.6.1 Complaints-Based Trust

One of the first reputation management algorithms for the P2P systems was proposed by Aberer and Despotovic [5]. This system is based solely on negative feedback given by peers when they are not satisfied by the files received from another peer. The system works on the assumption that a low probability of cheating in a community makes it harder to hide malicious behavior.

Let P denote the set of all peers in the network and B be the behavioral data consisting of trust observations $t(q, p)$ made by a peer $q \in P$ when it interacts with a peer $p \in P$. We can assess the behavioral data of a specific peer p based on the set

$$B(p) = \{t(p, q) \ or \ t(q, p) \mid q \in P\} \tag{85.5}$$

In this manner, the behavior of a peer takes into account not only all reports made *about* p but also all reports made *by* p. In a decentralized system a peer q does not have access to the global data $B(p)$ and B. Hence it relies on direct evidence as well as indirect evidence from a limited number of witnesses $r \in W_q \subset P$:

$$B_q(p) = \{t(q, p) \mid t(q, p) \in B\} \tag{85.6}$$

$$W_q(p) = \{t(r, p) \mid t(r, p) \in B \wedge r \in P\} \tag{85.7}$$

However, the witness r itself may be malicious and give false evidence. Aberer and Despotovic assume that peers only lie to cover their own bad behavior. If peer p is malicious and cheats peer q, q will file a complaint against it. If p also files a complaint against q at the same time it could be difficult to find out who is the malicious peer. However, if p keeps acting maliciously it will become easy to detect it since there will be a lot of complaints filed from peer r about a set of good peers and a lot of complaints filed from these good peers all about peer p. Based on this, the reputation of p can be calculated as

$$T(p) = |\{t(p, q) \mid q \in P\}| \times |\{t(q, p) \mid q \in P\}| \tag{85.8}$$

Aberer and Despotovic proposed a decentralized storage system called *P-Grid* to store reputation information. Each peer p can file a complaint about another peer q at any time as follows:

$$\mathit{insert}(a_i, t(p, q), \mathit{key}(p)) \ \mathit{and} \ \mathit{insert}(a_j, t(p, q), \mathit{key}(q))$$

where a_i and a_j are two arbitrary agents. Insertions are made on the keys of both p and q since the system stores complaints both by and about a given peer.

Assuming that an agent is malicious with probability π and that an error rate of ε is tolerable, then a peer p will need to receive r replicas of the same data satisfying $\pi^r < \varepsilon$ to ensure that the error rate is not exceeded. If a simple majority rule is employed, the total number of queries for each trust verification will never exceed $2r + 1$.

A peer p making s queries will obtain a set

$$W = \{(cr_i(q), cf_i(q), f_i, a_i) \mid i = 1, \ldots, w\}$$

where w is the number of different witnesses found, a_i the identifier of the ith witness, f_i the frequency with which witness a_i is found and $s = \sum_{i=1}^{w} f_i$. $cr_i(q)$ and $cf_i(q)$ are the complaints about q and filed by q, respectively, as reported by witness a_i. Different witnesses are found with different frequencies and the ones which are found less frequently have probably been found less frequently also when complaints were filed. So it is necessary to normalize $cr_i(q)$ and $cf_i(q)$ using frequency f_i in this way[4]:

$$cr_i^{norm}(q) = cr_i(q)\left(\frac{s - f_i}{s}\right)^s \tag{85.9}$$

$$cf_i^{norm}(q) = cf_i(q)\left(\frac{s - f_i}{s}\right)^s \tag{85.10}$$

Each peer p can keep a statistics of the average number of complaints filed (cf_p^{avg}) and received (cr_p^{avg}) and can determine if a peer q is trustworthy basing on the information returned from an agent i using this algorithm:

Algorithm (Trust Assessment Using Complaints)

```
$decide(cr_i^{norm}(q), cf_i^{norm}(q)) =$
if
$cr_i^{norm}(q) cf_i^{norm}(q) \leq \left(\frac{1}{2} + \frac{4}{\sqrt{cr_p^{avg} cf_p^{avg}}}\right)^2 cr_p^{avg} cf_p^{avg}$
then return 1; else return –1.
```

This algorithm assumes that if the total number of complaints received exceeds the average number of complaints by a large amount, the agent must be malicious.

85.6.2 EigenTrust

Kamvar et al. [7] presented a distributed algorithm for the computation of the trust values of all peers in the network. Their algorithm is inspired by the PageRank algorithm used by Google and assumes that trust is transitive. A user weighs the trust ratings it receives from other users by the trust it places in the reporting users themselves. Global trust values are then computed in a distributed fashion by updating the trust vector at each peer using the trust vectors of neighboring peers. They show that trust values asymptotically approach the eigenvalue of the trust matrix, conditional on the presence of pretrusted users that are always trusted.

A peer i may rate each transaction with peer j as positive ($tr(i, j) = 1$) or negative ($tr(i, j) = -1$). The local trust value at i for j can then be computed by summing the ratings of individual transactions:

$$s_{ij} = \sum tr_{(}i, j) \tag{85.11}$$

Local trust values are normalized as follows:

$$c_{ij} = \frac{\max(s_{ij}, 0)}{\sum_l \max(s_{il}, 0)} \tag{85.12}$$

to keep them between 0 and 1.

[4] Note that the equation below corrects the one presented in the original paper.

To aggregate normalized local trust values, trust values furnished by acquaintances are weighted by the trust a peer has in the acquaintances themselves:

$$t_{ik} = \sum_j c_{ij}c_{jk} \qquad \text{(note that } \sum\nolimits_j t_{ij} = 1\text{)} \tag{85.13}$$

where t_{ik} is the trust a peer i places in peer k based on the opinion of its acquaintances.

If we define $\vec{t_i}$ the vector containing the values t_{ik} and C the matrix containing the values c_{ij}, we get $\vec{t_i} = C^T \ \vec{c_i}$. The preceding expression only takes into account opinions of a peer's acquaintances. To get a wider view a peer may ask its friends' friends and so on:

$$\vec{t} = (C^T)^n \ \vec{c_i} \tag{85.14}$$

Kamvar et al. show that if n is large, peer i can have a complete view of the network and the trust vector $\vec{t_i}$ will converge to the same vector for every peer i (the **left principal eigenvector of** C), conditional on C being irreducible and aperiodic.

They further add three practical issues to this simple algorithm. If there are peers that can be trusted a priori the algorithm can be modified to take advantage of this. They define p_i as $\frac{1}{|P|}$ (where P is the set containing the pretrusted peers) if i is a pretrusted peer, and 0 otherwise. In presence of malicious peers, using an initial trust vector of $\vec{p}$ instead of $\vec{e}$ generally ensures faster convergence. If a peer i has never had any interaction with other peers, instead of being left undefined, c_{ij} can be defined as

$$c_{ij} = \begin{cases} \dfrac{\max(local_{ij}, 0)}{\sum_l \max(local_{il}, 0)} & \text{if } \sum_l \max(local_{il}, 0) \neq 0 \\ p_j & \text{otherwise} \end{cases} \tag{85.15}$$

To prevent malicious collectives from subverting the system, Kamvar et al. further modify the trust vector calculation on the k+1th iteration to:

$$\vec{t}^{\,k+1} = (1-a)C^T \ \vec{t}^{\,k} + a \ \vec{p} \tag{85.16}$$

where a is some constant less than 1. This ensures that at each iteration some of the trust must be placed in the set of pretrusted peers thus reducing the impact of the malicious collective.

Hence, given A_i the set of peers which have downloaded files from peer i and B_i the set of peers from which peer i has downloaded files, each peer i executes the following algorithm:

Algorithm (Distributed EigenTrust)

```
Query all peers $j \in A_i$ for $c_{ji}t^{(0)} = c_{ji}p_j$;
repeat
$t_i^{(k+1)} = (1-a)\left(c_{1i}t_1^{(k)} + c_{2i}t_2^{(k)} + \cdots + c_{ni}t_n^{(k)}\right) + ap_i$;
send $c_{ij}t_i^{(k+1)}$ to all peers $j \in B_i$;
wait for $c_{ji}t_j^{(k+1)}$ from all peers $j \in A_i$;
$\delta = \left|t^{(k+1)} - t^{(k)}\right|$;
until $\delta < \varepsilon$;
```

Each peer thus obtains the same global trust value matrix for all other peers in the network. Kamvar et al. further describe a DHT-based solution to anonymously store multiple copies of the trust value for a given peer at several *score managers*. This alleviates the problem caused by malicious peers reporting false trust values for themselves to other peers to subvert the system.

85.6.3 PeerTrust

In PeerTrust, Xiong and Liu [8] define five factors used to compute the trustworthiness of a peer. These are (1) feedback obtained from other peers, (2) scope of feedback such as number of transactions, (3) credibility of feedback source, (4) transaction context factor to differentiate between mission-critical and noncritical transactions, and (5) community context factor for addressing community-related characteristics and vulnerabilities.

Given a recent time window, let $I(u, v)$ denote the total number of transactions between peer u and v, $I(u)$ the total number of transactions of peer u, $p(u, i)$ the peer u's partner in its ith transaction, $S(u, i)$ the normalized amount of satisfaction peer $p(u, i)$ receives from u in this transaction, $Cr(v)$ the credibility of peer v, $TF(u, i)$ the adaptive transaction context factor for peer u's ith transaction, and $CF(u)$ the u's community context factor. Then, the trust value of peer u denoted by T(u) is

$$T(u) = \alpha * \sum_{i=1}^{I(u)} S(u, i) * Cr(p(u, i) * TF(u, i) + \beta * CF(u) \qquad (85.17)$$

where α and β are weight factors.

In their experimental study, Xiong and Liu turn off the transaction context factor and the community context factor and use two credibility metrics. The first is based on the trust value of the reporting peer (similar to EigenTrust) while the second is based on the similarity between the reporting peer and the recipient of the trust information. They further propose using a PKI-based scheme and data replication to increase the security and reliability of their system.

85.6.4 ROCQ

Garg et al. [9,45] proposed a scheme that combines local opinion, credibility of the reporter and the quality of feedback to compute the reputation of a peer in the system. In their scheme direct evidence in the form of local opinion is reported to score managers which are chosen using distributed hash tables.

The reputation R_{mj} of user j at score manager m is

$$R_{mj} = \frac{\sum_i O_{ij}^{avg} C_{mi} Q_{ij}}{\sum_i C_{mi} Q_{ij}} \qquad (85.18)$$

where C_{mi} is the credibility of user i according to m, O_{ij}^{avg} i's average opinion of j and Q_{ij} the associated quality value reported by i.

The quality value of an opinion depends on the number of transactions on which the opinion is based and the consistency with which the transaction partner has acted. Thus an opinion is of greater quality when the number of observations on which it is based is larger and when the interactions have been consistent (resulting in a smaller variance). When the number of observations is high but they do not agree with each other, the quality value is lower.

The credibility of a user is based on direct evidence only and is not shared with other users. This prevents the recursion problem of calculating the third-order reputation and so on. The credibility is based upon the agreement of the reported opinion with the group consensus as reflected in the reputation value and is updated after every report received. The precise formula for adjusting the credibility of user i by user m is

$$C_{mi}^{k+1} = \begin{cases} C_{mi}^{k} + \frac{(1-C_{mi}^{k})\cdot Q_{ij}}{2}\left(1 - \frac{|R_{mj} - O_{ij}^{avg}|}{s_{mj}}\right) & \text{if } |R_{mj} - O_{ij}^{avg}| < s_{mj} \\ C_{mi}^{k} - \frac{C_{mi}^{k}\cdot Q_{ij}}{2}\left(1 - \frac{s_{mj}}{|R_{mj} - O_{ij}^{avg}|}\right) & \text{if } |R_{mj} - O_{ij}^{avg}| > s_{mj} \end{cases} \qquad (85.19)$$

where C_{mi}^{k} is the credibility of user i after k reports to user m, O_{ij}^{avg} the opinion being currently reported by user i, Q_{ij} the associated quality value, R_{mj} the aggregated reputation value that user m computed for

j, and s_{mj} the standard deviation of all the reported opinions about user j. In this way, if a reporting user is malicious, its credibility rating is gradually reduced since its opinion does not match that of the community as a whole.

The authors also propose combining both direct and indirect evidence to create reputation. They proposed a threshold number of interactions between two users below which users will rely on the global reputation of their prospective partner and above which they would rely on first-hand evidence only. This eliminates the problem of sparsity of data while at the same time allowing for reputation to be tailored according to personal experience.

85.6.5 Propagation of Trust and Distrust

A number of mathematical approaches to propagating trust [46,47] and distrust [48] have been proposed. In particular, Guha et al. gave a number of models of atomic (single-step) propagation. Let B be a belief matrix whose *ij*th element signifies i's belief in j. B can be composed of either the trust matrix (T_{ij} is i's trust in j) or both the trust and the distrust (D_{ij} is i's distrust in j) matrices (say $B_{ij} = T_{ij} - D_{ij}$). Then, atomic propagation of trust takes place by (1) *direct propagation* (matrix operator[5]: **B**) : assumes that trust is *transitive* so if i trusts j and j trusts k then we can infer that i trusts k, (2) *co-citation* ($B^T B$): if both i and j trust k and if i trusts l, then j also trusts l, (3) *transpose trust* (B^T): assumes that trust is *reflexive* so that if i trusts j then trusting j should imply trusting i, and (4) *trust coupling* (BB^T): if i and j trust k, then trusting i should also imply trusting j. They then go on to propagate trust using a combined matrix that gives weights to the four propagation schemes:

$$C_{B,\alpha} = \alpha_1 B + \alpha_2 B^T B + \alpha_3 B^T + \alpha_4 BB^T \tag{85.20}$$

Thus, applying the atomic propagations a fixed number of times, new beliefs can be computed. In a limited set of experiments, they study the prediction performance of their algorithm on real data and find that the best performance comes with one-step propagation of distrust while trust can be propagated repeatedly.

85.7 Conclusions and Future Work

The area of reputation systems remains fertile for future research. Initial research in this area has focused on applying lessons from diverse fields such as evolutionary biology and game theory to computer science. In particular game-theoretic models such as the iterated prisoner's dilemma were adapted to virtual communities. However this analysis is limited by not considering the presence of irrational (malicious) players in the community.

Simultaneously, several new reputation schemes have been proposed. These schemes typically propose a new trust model followed by experimental simulation. However it is difficult to compare the schemes side-by-side as each scheme makes its own assumptions about the interaction model, modes of maliciousness and levels of collusion among malicious peers, not to mention widely varying experimental parameters such as the number of peers in the system and the proportion of malicious peers.

More recently, there has been some work on analyzing systems in the presence of malicious peers. For instance, Mundinger and Le Boudec [49] analyze the robustness of their reputation system in the presence of liars and try to find the critical-phase transition point where liars start impacting the system.

As the interest in virtual communities, particularly self-organizing communities grows, we are likely to see a lot more research on the various facets of this topic.

Note: This work was supported in part by the European Commission through the FET project CASCADAS.

[5] The matrix operator when applied to a belief matrix would yield a new matrix indicating inferred trust.

References

[1] Zucker, L., Production of trust: institutional sources of economic structure 1840–1920, *Res. Org. Behavior*, 8(1), 53, 1986.

[2] Kirsner, S., Catch me if you can, *Fast Company*, http://www.fastcompany.com/magazine/73/kirsner.html, retrieved August 13th, 2005.

[3] Atif, Y., Building trust in e-commerce, *IEEE Internet Comput.*, 6(1), 18–24, 2002.

[4] Douceur, J., The sybil attack, *Proc. Int. Peer To Peer Systems Workshop*, 2002.

[5] Aberer, K. and Despotovic, Z., Managing trust in a peer-2-peer information system, *CIKM*, 2001, 310.

[6] Damiani, E., di Vimercati, S. D. C., Paraboschi, S., and Samarati, P., Managing and sharing servents' reputations in P2P systems, *IEEE Trans. Data and Knowledge Eng.*, 15(4), 840, 2003.

[7] Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H., The eigentrust algorithm for reputation management in P2P networks, *Proc. Int. Conf. on World Wide Web*, 2003, p. 640.

[8] Xiong, L. and Liu, L., PeerTrust: supporting reputation-based trust in peer-to-peer communities, *IEEE Trans. Data and Knowledge Eng., Special Issue on Peer-to-Peer Based Data Manage.*, 16(7), 843, 2004.

[9] Garg, A., Battiti, R., and Costanzi, G., Dynamic self-management of autonomic systems: the reputation, quality and credibility (RQC) scheme, *Int. Workshop on Autonomic Communication*, 2004.

[10] Buchegger, S. and Le Boudec, J.-Y., A robust reputation system for P2P and mobile ad-hoc networks, *Proc. Workshop on the Economics of Peer-to-Peer Systems*, 2004.

[11] Trivers, R. L., The evolution of reciprocal altruism, *Quart. J. Biol.*, 46, 35, 1971.

[12] Gintis, H., Smith, E. A., and Bowles, S., Costly signalling and cooperation, *J. Theor. Biol.*, 213, 103, 2001.

[13] Fehr, E. and Gächter, S., Altruistic punishment in humans, *Nature*, 415(6868), 137, 2002.

[14] Axelrod, R., *The Evolution of Cooperation*, Basic Books, New York, 1984.

[15] Jackson, M., Mechanism Theory, 2003, `http://instruct1.cit.cornell.edu/courses/econ669/jackson2.pdf`, retrieved August 15th, 2005.

[16] Akerlof, G. A., The market for 'lemons': quality uncertainty and the market mechanism, *Quart. J. Econ.*, 84(3), 488, 1970.

[17] Morselli, R., Katz, J., and Bhattacharjee, B., A game-theoretic framework for analyzing trust-inference protocols, *Proc. Workshop on the Economics of Peer-to-Peer Systems*, 2004.

[18] Avery, C., Resnick, P., and Zeckhauser, R., The market for evaluations, *Am. Econ. Rev.*, 89(3), 564, 1999.

[19] Resnick, P., Zeckhauser, R., Swanson, J., and Lockwood, K., The value of reputation on eBay: a controlled experiment, *Proc. ESA Conf.*, 2002.

[20] Kakade, S. M., Kearns, M., Ortiz, L. E., Pemantle, R., and Suri, S., Economic properties of social networks, in *Advances in Neural Information Processing Systems 17*, Saul, L. K., Weiss, Y., and Bottou, L., Eds., MIT Press, Cambridge, MA, 2005.

[21] Resnick, P., Zeckhauser, R., Friedman, E., and Kuwabara, K., Reputation systems: facilitating trust in Internet interactions, *CACM*, 43(12), 45, 2000.

[22] Boykin, P. O. and Roychowdhury, V., Personal email networks: an effective anti-spam tool, *IEEE Comput.*, 38(4), 61, 2005.

[23] Zimmerman, P. and Philip, R., *The Official PGP User's Guide*, MIT Press, Cambridge, MA, 1995.

[24] Callas, J., Donnerhacke, L., Finney, H., and Thayer, R., RFC 2440—OpenPGP message format, 1998.

[25] Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., and Riedl, J., GroupLens: an open architecture for collaborative filtering of netnews, *Proc. ACM Conf. on Computer Supported Cooperative Work*, 1994, p. 175.

[26] Breese, J., Heckerman, D., and Kadie, C., Empirical analysis of predictive algorithms for collaborative filtering, *Proc. Uncertainty in Artificial Intelligence Conf.*, 1998, p. 43.

[27] Dellarocas, C., Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior, *Proc. Conf. on Electronic Commerce*, 2000, p. 150.

[28] Hardin, G., The tragedy of the commons, *Science*, 162, 1243, 1968.

[29] Saroiu, S., Gummadi, P., and Gribble, S., A measurement study of peer-to-peer file sharing systems, 2002.

[30] Granovetter, M., Economic action and social structure: the problem of embeddedness, *Am. J. Sociol.*, 91, 481, 1985.

[31] Fudenburg, D. and Tirole, J., *Game Theory*, MIT Press, Cambridge, MA, 1991.

[32] Ahlswede, R., Cai, N., Li, S.-Y. R., and Yeung, R. W., Network information flow, *IEEE Trans. Inf. Theor.*, 46(4), 1204, 2000.

[33] Biersack, E., Rodriguez, P., and Felber, P., Performance analysis of peer-to-peer networks for file distribution, *Proc. Workshop on Quality of Future Internet Services*, 2004.

[34] Byers, J. W., Considine, J., Mitzenmacher, M., and Rost, S., Informed content delivery across adaptive overlay networks, *IEEE/ACM Trans. Networking*, 12(5), 767, 2004.

[35] Cohen, B., Incentives build robustness in BitTorrent, *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[36] Zacharia, G., Moukas, A., and Maes, P., Collaborative reputation mechanisms in electronic marketplaces, *Proc. Hawaii Int. Conf. on Systems Sciences*, 8, 1999, 8026.

[37] Dellarocas, C., Mechanisms for coping with unfair ratings and discriminatory behavior in online reputation reporting systems, *Proc. Int. Conf. on Information Systems*, 2000, p. 520.

[38] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S., A scalable content addressable network, *Proc. SIGCOMM Conf.*, 2001, p. 161.

[39] Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H., Chord: a scalable peer-to-peer lookup service for internet applications, *Proc. SIGCOMM Conf.*, 2001, p. 149.

[40] Jurca, R. and Faltings, B., An incentive compatible reputation mechanism, *Proc. IEEE Conf. on E-Commerce*, 2003.

[41] Berg, J., Dickhaut, J., and McCabe, K., Trust, reciprocity and social history, *Games Econ. Behav.*, 10, 122, 1995.

[42] Mui, L., Mohtashemi, M., and Halberstadt, A., Notions of reputation in multi-agents systems: a Review, *Proc. Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, 2002, p. 280.

[43] Sabel, M., Garg, A., and Battiti, R., WikiRep: digital reputations in collaborative applications, *Proc. AICA Congress*, 2005.

[44] Friedman, E. and Resnick, P., The social cost of cheap pseudonyms, *J. Econ. Manage. Strategy*, 10(1), 173–199, 2001.

[45] Garg, A., Battiti, R., and Cascella, R., Reputation management: experiments on the robustness of ROCQ, *Proc. Int. Symp. on Autonomous Decentralized Systems*, 2005, p. 725.

[46] Richardson, M., Agarwal, R., and Domingos, P., Trust management for the semantic web, *Proc. Int. Semantic Web Conf.*, 2003, p. 351.

[47] Golbeck, J., Parsia, B., and Hendler, J., Trust networks on the semantic web, *Proc. Cooperative Intelligent Agents*, 2003.

[48] Guha, R., Raghavan, P., Kumar, R., and Tomkins, A., Propagation of trust and distrust, *Proc. WWW 2004*, 2004, p. 403.

[49] Mundinger, J. and Le Boudec, J. Y., The impact of liars on reputation in social networks, *Proc. Social Network Analysis: Advances and Empirical Applications Forum*, 2005.

86

Color Quantization

Zhigang Xiang

Queens College of the City University of New York

86.1 Introduction

Quantization is originally defined as the digitization of a continuous signal. The signal may represent a monochromatic tone varying between black and white, and be measured by its intensity. The task of quantization is to map the signal's intensity values into a series of gray levels (e.g., 256 levels from black to white along the gray axis). The signal may also carry chromatic information, with multiple attributes that characterize the signal within a multidimensional color space (e.g., $\boldsymbol{c} = (r, g, b)$ for the RGB space). In this case quantization amounts to mapping color points onto a grid that discretizes the color space (e.g., 256 levels in each dimension).

The advent of digital image processing gives rise to the need to reduce the number of colors that are present in an image (e.g., a 24-bit image with tens of thousands of colors from a $256 \times 256 \times 256$ grid) by reassigning pixels to a smaller set of grid values (e.g., 256 for the 8-bit lookup table representation). Thus the problem of color quantization or color image quantization evolves to become the problem of remapping already quantized image colors. This serves a variety of purposes as quantized images have lower memory consumption, allow speedier transmission, and place lesser demand on processing hardware.

Let G be the set of grid values (i.e., possible colors) and $C = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n\} \subseteq G$ be the set of n original colors in an image. The problem of quantizing the image into k final colors, which are now referred to as quantized colors and denoted by $Q = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_k\} \subset G$, can be stated as finding Q along with a mapping from C to Q, where Q is generally not a subset of C, and is often called the color map or color palette (or codebook, where each quantized color is a code word) for the image. On the other hand, considering that this mapping from C to Q implies that pixels with the same original color are destined to have the same quantized color, which is usually the case but rather restrictive, we may adopt a more general characterization by defining the quantization task as finding Q along with an assignment of a quantized color to each pixel.

Quantization inevitably introduces distortion. An ideal quantization algorithm should make the quantized image look as close to the original and exhibit as few objectionable artifacts as possible. Aside from the apparent difficulty in quantifying this objective due to its subjective nature, we also face a couple of complicating factors that are relatively immune to variations in individual judgment. Together they make color quantization a good candidate for approximation algorithms and heuristics.

The first complicating factor stems from the fact that the sensation of color is a psychophysical phenomenon, which results from physical stimulus in the form of visible light entering our visual system.

Ideally, distances in a color space where the physical stimulus is measured are proportional to the perceived differences by the viewer—such a color space is often referred to as being perceptually uniform. We can then equate the task of minimizing the distances between original and quantized colors (quantization errors) with the task of minimizing the visible discrepancies between those colors. However, the prevailing RGB color model for image representation is far from being perceptually uniform, and the visualization of an RGB image is affected by the physical characteristics of the display or printing device [1,2]. A numerical displacement to an original color may result in a minute color change that is hardly visible when the displacement occurs in some parts of the RGB color space, but manifests as a clearly noticeable color shift when it takes place in other regions. Although we may map RGB colors into another color space that is device-independent and significantly more uniform to carry out the quantization task [3,4], it is still work-in-progress to find a color model that is truly perceptually uniform.

The second complicating factor comes from the context-dependent nature of the quantization errors that are considered visually offensive by an average observer. An image is not a simple collection of isolated individual data points in the eyes of the viewer, its quality needs to be judged with all pixels taken as a sophisticated whole. Even if we conduct quantization in a color space that is perfectly perceptually uniform (and we have perfectly calibrated display and printing devices), the resulting distribution of quantization errors (minimized one way or another without regard to context) does not necessarily guarantee the minimization of their visual offensiveness. For instance, consider the impact of a certain visible shift of the colors that are referred to as the skin tones in a quantized image. The shift may very well be acceptable when the colors depict ordinary objects such as a piece of fabric or flowers (the objects are quite likely to look just fine in the shifted colors); however, the same shift can be rather objectionable when the colors happen to portray a human face (proper skin tones are indicative of the subject's well being). Furthermore, there are such factors as the spatial averaging effect of color stimuli from adjacent pixels through our visual pathway—much like the spatial averaging of subpixels that enables RGB display devices to work, and the phenomenon of simultaneous contrast [5] that affects our perception of color when differently colored patches are viewed in each other's presence—the two small disks in Figure 86.1 have exactly the same color but the one on the right looks brighter because it is surrounded by a dark ring. There are also many other aspects (ranging from psychophysical to cognitive) of visual information processing that are not yet fully understood.

Given the complexity of the color quantization problem, it should come as no surprise that existing color quantization methods tend to focus on limited aspects of the problem and have relatively confined goals. Based on the scope of the information that is used in the construction of the color map, we may divide these methods into three broad categories (see Figure 86.2): (1) image-independent methods that determine a universal set of quantized colors without regard to any specific image; (2) image-dependent, context-free methods that take into account the actual colors that appear in the input image as well as the frequency (i.e., number of occurrences or pixel population) of each of those colors, and typically focus on the minimization of the numerical discrepancies between original and quantized colors based on

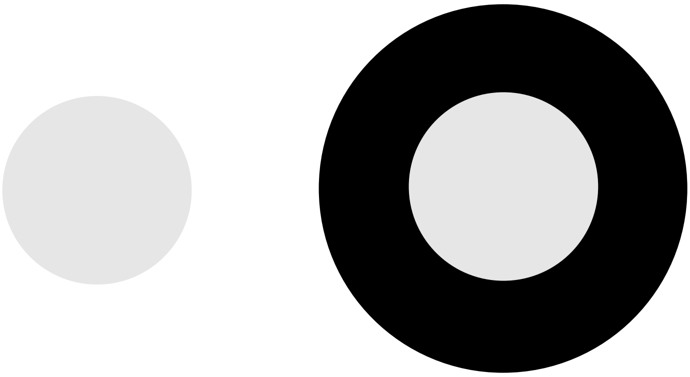


FIGURE 86.1 Demonstration of simultaneous contrast.

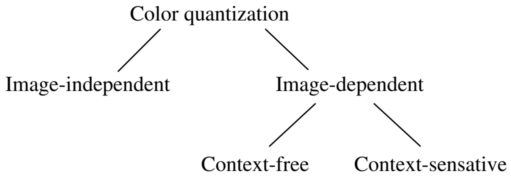


FIGURE 86.2 Categorization of color quantization methods.

certain statistical criteria; and (3) image-dependent, context-sensitive methods that make use of additional contextual information beyond original colors and their frequencies that can be derived from the input image (e.g., spatial relationship between the pixels) as heuristics to help better restrain visible quantization artifacts.

Many algorithms are essentially color space-neutral by design, leaving it a separate issue for the pixel colors to be represented in an appropriate color space. However, the RGB color space is frequently used in experimental implementation.

Three common options exist for the mapping of original colors to quantized colors, or more broadly, the assignment of quantized colors to pixels: (1) replace each original color with the closest counterpart in the color map (mostly image-independent methods); (2) replace each original color with a specific quantized color whose association with the original color has already been determined during the construction of the color map (mostly image-dependent methods), and (3) make use of such techniques as error diffusion [6–9] to select the appropriate quantized color for each pixel. The latter option helps to smooth out some of the visible quantization artifacts, with the trade-off being that it may also degrade sharp edges and fine details.[1]

86.2 Color Spaces for Quantization

The CIELUV and CIELAB are two prominent candidate spaces that are derivatives of the CIE 1931 XYZ color model, which is device-independent but nonuniform in terms of perceived color differences [10]. Both are defined as nonlinear transformations of XYZ with respect to a reference white point, which may be the standard illuminant D50 for reflective reproduction, with XYZ coordinates being (0.9642, 1.0, 0.8249), or D65 for emissive display, with XYZ coordinates being (0.9504, 1.0, 1.0889).

The CIELUV or CIE 1976 $L^*u^*v^*$ color space is defined by

$$\begin{aligned}
L^* &= \begin{cases} 116\left(\dfrac{Y}{Y_w}\right)^{1/3} - 16, & \left(\dfrac{Y}{Y_w}\right) > 0.008856 \\ 903.3\left(\dfrac{Y}{Y_w}\right), & \text{otherwise} \end{cases} \\
u^* &= 13L^*(u' - u'_w) \\
v^* &= 13L^*(v' - v'_w)
\end{aligned}$$

where

$$u' = \frac{4X}{X + 15Y + 3Z} \quad u'_w = \frac{4X_w}{X_w + 15Y_w + 3Z_w}$$

$$v' = \frac{9Y}{X + 15Y + 3Z} \quad v'_w = \frac{9Y_w}{X_w + 15Y_w + 3Z_w}$$

X_w, Y_w, and Z_w are determined from the reference white point.

[1] Edge detection has been used to suppress error diffusion across edges to preserve image sharpness [28]; and edge enhancement may be incorporated into the error diffusion process [54].

The CIELAB or CIE 1976 $L^*a^*b^*$ color space is defined by

$$L^* = \begin{cases} 116\left(\frac{Y}{Y_w}\right)^{1/3} - 16, & \left(\frac{Y}{Y_w}\right) > 0.008856 \\ 903.3\left(\frac{Y}{Y_w}\right), & \text{otherwise} \end{cases}$$

$$a^* = 500\left(f\left(\frac{X}{X_w}\right) - f\left(\frac{Y}{Y_w}\right)\right)$$

$$b^* = 200\left(f\left(\frac{Y}{Y_w}\right) - f\left(\frac{Z}{Z_w}\right)\right)$$

where

$$f(t) = \begin{cases} t^{1/3} & t > 0.008856 \\ 7.787t + \frac{16}{116} & \text{otherwise} \end{cases}$$

X_w, Y_w, and Z_w are determined from the reference white point.

The L^* component in both trivariant models is designed to carry luminance and the remaining two specify chrominance. Although often referred to as being perceptually uniform, these two color spaces still depart from uniformity over the visible gamut, with variations that may reach as high as 6:1 when color differences are measured by Euclidian distances $\Delta E_{uv} = \sqrt{\Delta L^{*2} + \Delta u^{*2} + \Delta v^{*2}}$ and $\Delta E_{ab} = \sqrt{\Delta L^{*2} + \Delta a^{*2} + \Delta b^{*2}}$ [11,12]. An improved color difference formula was introduced in 1994 [11]:

$$\Delta E^*_{94} = \sqrt{\left(\frac{\Delta L^*}{k_L S_L}\right)^2 + \left(\frac{\Delta C^*_{ab}}{k_C S_C}\right)^2 + \left(\frac{\Delta H^*_{ab}}{k_H S_H}\right)^2}$$

which is based on using polar coordinates to address color points in the CIELAB space in terms of perceived lightness L^*, chroma $C^*_{ab} = \sqrt{a^{*2} + b^{*2}}$, and hue angle $H^*_{ab} = \tan^{-1}(\frac{b^*}{a^*})$. Standard reference values for the formula are $k_L = k_C = k_H = 1$, $S_L = 1$, $S_C = 1 + 0.045C^*_{ab}$, and $S_H = 1 + 0.015C^*_{ab}$.

The conversion between RGB (assumed to be linear without γ correction for cathode ray tubes) and XYZ may be carried out by the following standard transformation, with white illuminant D65:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.0651 & -1.3942 & -0.4761 \\ -0.9690 & 1.8755 & 0.0415 \\ 0.0679 & -0.2290 & 1.0698 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

86.3 Image-Independent Quantization

In contrast with the image-dependent methods that choose quantized colors in an image-specific fashion, quantization may also be carried out on an image-independent basis, with a fixed/universal palette for all images. These image-independent methods enjoy high computational efficiency since they avoid the need to analyze each original image to determine the quantized colors for that image. And there is no overhead for the storage and transmission of the individualized color map for each image. The trade-off is that a set of quantized colors that are specifically tailored to the distribution of the original colors in a given image tends to do a better job in approximating those original colors and lowering quantization errors.

Uniform quantization. In this approach we preselect k colors that are uniformly distributed in the chosen color space (preferably perceptually uniform). A ready example would be the $6 \times 6 \times 6$ browser/web-safe palette, with integer values 0, 51, 102, 153, 204, and 255 for each primary for a total of 216 RGB colors [13]. The quantization of an image now entails mapping each pixel to a preselected color (e.g., one that is the closest to the pixel's original color).

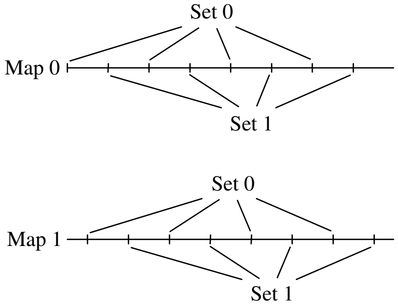


FIGURE 86.3 Trellis-coded quantization.

An easy and fast implementation of uniform quantization involves the truncation of a few least-significant bits from each component of an original color, rounding the original color down to a quantized color. For example, we may truncate 3 bits from each component of a 24-bit RGB color to arrive at its counterpart in a set of 32 × 32 × 32 quantized colors. Alternatively, aiming to better preserve luminance (a key ingredient that conveys details) and taking hint from the standard formula for computing luminance from RGB values: $Y = 0.299R + 0.587G + 0.114B$, we may truncate 3 bits from the red component, 2 bits from the green component, and 4 bits from the blue component to partially compensate for the nonuniform nature of the RGB color space.

This bit-cutting technique effectively places all quantized colors below the maximum intensity level in each dimension of the color space, and causes a downward shift in intensity (as well as hue shift) across the entire image. These are often unacceptable when a relatively high number of bits are truncated.

Trellis-coded quantization. Consider the case of uniform quantization using one byte for the direct encoding of pixel colors, for example, 3-3-2 for red-green-blue, we would have a rather coarse grid of 8 × 8 × 4 quantized colors. Now if we can "extend" the capacity of the limited number of bits used for each primary to specify intensity values at a higher resolution, we will be able to approach the effect of uniform quantization within a finer grid. This is made possible by the application of the Viterbi algorithm [14,15] in trellis-coded quantization [16–18].

Take, for example, the encoding of one of the primaries with $x = 3$ bits, which normally yields $2^x = 2^3 = 8$ intensity levels. In contrast, we may have two color maps (see Figure 86.3), each of which consists of eight equally spaced intensity levels. The values in one map can be obtained by offsetting the values in the other map by half the distance between two adjacent levels. We further partition the intensity values in each map into two subsets. Given a specific map, only $x = 3$ bits are necessary to identity one of the two subsets (1 bit needed) and the particular intensity level within the chosen subset ($x - 1 = 2$ bits needed). Operating as a finite state machine, described by a trellis, the algorithm uses the bit that identifies the subset within the current map to determine the next state of the trellis, which in turn determines the choice of color map for the next input bit-string. This approximates the effect of quantizing with $2^{x+1} = 2^4 = 16$ intensity levels.

Sampling by Fibonacci lattice. Unlike scalar values on the gray axis, points in a multidimensional color space do not lend themselves to easy manipulation and ordering. In this variation of uniform quantization [19], the universal color palette is constructed by sampling within a series of cross planes along the luminance axis of the CIELAB color space. Each cross plane is a complex plane centered at the luminance axis, and sample points z_j in the plane (with equal luminance) are determined by the Fibonacci spiral lattice (see Figure 86.4):

$$z_j = j^\delta e^{i\theta}, \quad \theta = 2\pi j\tau + \alpha_0$$

where parameter δ controls the radial distribution of the points (higher value produces greater dispersion), τ determines the overall pattern of distribution (a Markoff irrational number yields the most uniform distribution), and α_0 denotes an initial angle of rotation to better align the sample points with the boundaries of the color space.

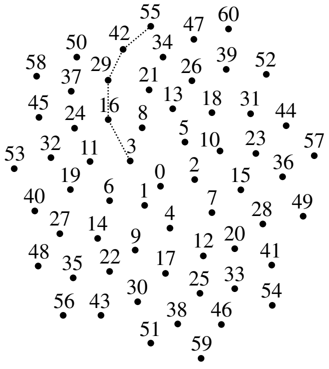


FIGURE 86.4 Points on the Fibonacci spiral lattice.

The values $\delta = 0.5$ and $\tau = \frac{\sqrt{5}-1}{2}$ (the golden mean) are used in implementation, along with an additional scaling factor to help adjust the sample points' coverage of the color space within each cross plane. To produce a universal palette of a certain size, the number of luminance levels and the number of sample points per level need to be carefully chosen, and the set of luminance values be determined through image-dependent quantization of luminance values from a large set of training images.

A unique aspect of this sampling method comes from the Fibonacci lattice. Each sample point z_j in the spiral lattice is uniquely determined by its scalar index j, and two neighboring points are always some Fibonacci number apart in their indices. These plus other useful properties of the Fibonacci lattice make the resulting color palette amenable to fast quantization and ordered dither. In addition, a number of gray-scale image processing operations such as gradient-based edge detection can be readily applied to the quantized color images.

86.4 Image-Dependent, Context-Free Quantization

Image-dependent, context-free quantization methods select quantized colors based solely on original colors and their frequencies, without regard to the spatial relationship of the pixels and the context of the visual information that the image conveys. A basic strategy shared by numerous quantization algorithms in this category is to proceed in two steps. The first partitions the n original image colors into k disjoint clusters $S_1, S_2, \ldots, S_k$ based on a certain numerical criterion—this makes color quantization a part of the broader area of data clustering [20]; and the second computes a representative (i.e., a quantized color) for each cluster. The quantized image may then be constructed by recoloring each pixel with the representative of the cluster that contains the pixel's original color, or with the application of such techniques as error diffusion using the resultant color map.

Intuitively, these methods differ in how to balance two interrelated and competing objectives: the preservation of popular colors versus the minimization of maximum quantization error (see Figure 86.5). The former may be characterized as achieving an error-free mapping for the k most popular original colors; whereas the latter is the minimization of the upper bound for all $d(\boldsymbol{c}, \boldsymbol{q}_i)$, $1 \leq i \leq k$, where $\boldsymbol{c}$ is an original color in the ith cluster S_i, $\boldsymbol{q}_i$ the representative of S_i, and $d(\boldsymbol{c}, \boldsymbol{q}_i)$ the nonnegative quantization error, typically the Euclidean distance between $\boldsymbol{c}$ and $\boldsymbol{q}_i$.

A classic approach to strike a balance between the two objectives is to minimize the sum of squared errors $\sum_{1 \leq i \leq k} \sum_{\boldsymbol{c} \in S_i} P(\boldsymbol{c}) d^2(\boldsymbol{c}, \boldsymbol{q}_i)$ across the entire image, where $P(\boldsymbol{c})$ is the frequency (pixel population) of color $\boldsymbol{c}$. As an alternative, we may try to minimize the total quantization errors $\sum_{1 \leq i \leq k} \sum_{\boldsymbol{c} \in S_i} P(\boldsymbol{c}) d(\boldsymbol{c}, \boldsymbol{q}_i)$, which represents a lesser bias towards capping the maximum quantization error. Such statistical criteria can trace their origin to the quantization of a continuous-tone black-and-white signal [21,22], and have been proven to be NP-complete [23–26].

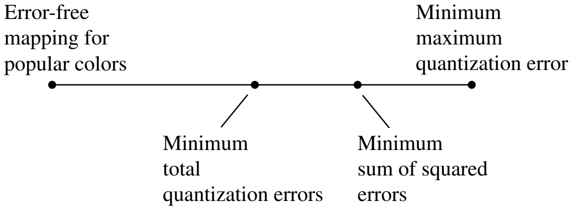

Error-free mapping for popular colors
Minimum maximum quantization error
Minimum total quantization errors
Minimum sum of squared errors

FIGURE 86.5 An intuitive scale for comparing statistical criteria.

Regardless of the operational principle (e.g., limiting the spatial extent of each cluster) for the clustering step of an approximation algorithm, the frequency-weighed mean of the original colors in each resulting cluster is almost always used as the cluster's representative, often called the cluster's centroid but sometimes referred to as the center of gravity (the two notions are equivalent in this context). This reflects a common consensus on minimizing intracluster variance.

The popularity algorithm. This early quantization method aims at the preservation of popular colors [27]. It creates a histogram of the colors in the original image and selects the k most popular ones as quantized colors. Pixels in other colors are simply mapped to the closest quantized colors, respectively. The advantage here is that relatively large and similarly colored image areas are kept little changed after quantization; however, smaller areas, some of which may carry crucial information (e.g., a uniquely colored signal light), can take on significant distortion as a result of their being mapped to popular colors.

An implementation technique that may alleviate this problem preprocesses the 24-bit original colors by truncating a few least-significant bits from each color component (i.e., performing a uniform quantization), effectively combining several popular colors that are very similar to each other into a single quantized color, thus allowing some of the less popular colors to be selected as quantized colors. This preprocessing step (e.g., 3-3-3 or 3-2-4 bit-cutting for red-green-blue) can also be used to achieve color reduction and to alter color granularity for other algorithms. The downside here is that bit-cutting itself can cause false contours to appear on smoothly shaded surfaces.

One may also avoid having several popular colors that are neighbors of each other as quantized colors by choosing one quantized color at a time, and artificially reducing the pixel count of the remaining colors in the vicinity of the chosen color (currently the most popular), with the reduction being based on a spherically symmetric exponential function in the form of $1 - e^{Kr^2}$ (note that this sets the pixel count of the chosen color to 0 so it will never be selected again in subsequent iterations), where r is the radius of the sphere that is centered at the chosen color and K an experimentally determined constant [28].

Detecting peaks in histogram. Instead of choosing the k most popular original colors for the color map, we may find peaks in the histogram and use colors at the peaks as quantized colors. The peaks may be identified by a multiscale clustering scheme based on discrete wavelet transform (DWT), where computational efficiency comes from carrying out three-dimensional DWT as a series of independent one-dimensional transforms followed by downsampling [29]. The quantizer can determine the value of k from the number of detected peaks, or it may be adjusted to produce a preset number of quantized colors.

Peano scan. In another technique to lessen the difficulty associated with multidimensional data processing, a recursively defined space-filling curve, referred to as a Peano curve, is used to traverse the space (e.g., the RGB color cube), creating a one-to-one mapping between points in space and their counterparts along the curve. Subject to the spatial relationships that are preserved by the mapping, certain spatially oriented operations may now be carried out along a single dimension. For example, since points close on the Peano curve are also close in space, given a specific color we may easily find some of its neighbors by searching along the curve [30,31]. The shortfall of this approach comes from the fact that points close in space are only likely, but not necessarily to be close on the curve.

The Median-cut algorithm. This two-step algorithm conducts a hierarchical subdivision of clusters that have high pixel populations, attempting to achieve an even distribution of pixels among the quantized colors [27]. We first fit a rectangular box over an initial cluster containing all original colors, and split the box into two with a plane that is orthogonal to its longest dimension to bisect the cluster in such a way that each new cluster is now responsible for half of the pixel population in the original cluster. The new clusters are then treated the same way repeatedly until we have k clusters. The criterion for selecting the next cluster to split is based on pixel count. By splitting the most popular cluster in each step, the algorithm will eventually produce k clusters each of which is responsible for roughly $1/k$ of the image's pixel population.

In comparison with the popularity algorithm, this alternative for resource distribution often brings about better quantization results. However, having the same number of pixels mapped to each quantized color does not necessarily lead to effective control of quantization errors.

The center-cut algorithm. As a variation to the median-cut algorithm, this method bisects a cluster at the midpoint of the longest dimension of its bounding box without regard to pixel population [32]. And it ranks candidate clusters for subdivision based on the longest dimension of their bounding boxes—the longest one is split first. These changes put more emphasis on restraining the spatial extent of the clusters, and do a better job in keeping grossly distinct colors from being grouped into the same cluster and mapped to the same quantized color.

Both the median-cut and the center-cut algorithms take a top-down approach to partitioning a single cluster into k clusters. Alternatively, we may follow a bottom-up strategy that merges the n original colors into the desired number of clusters. To this end the octree data structure [33] can be used to provide a predetermined hierarchical subdivision of the RGB color space for merging clusters.

Octree quantization. With the entire RGB color cube represented by the root node and each octant of the color cube by a child node descending from the root, an individual 24-bit RGB color corresponds to a leaf node at depth 8 of the octree [34]. Conceptually, once we populate an octree with pixel colors from an input image, we may start from the bottom of the octree (greatest depth) and recursively merge leaf nodes that have the same parent into the parent node, transforming the parent node into a leaf node at the next level, until we reduce the number of leaf nodes from n to k. Each remaining leaf node now represents a cluster of original colors that inhabit the spatial extent of the node.

In an actual implementation, only an octree structure with no more than k leaf nodes needs to be maintained, where each leaf node has a color accumulator and a pixel counter for the eventual calculation of its centroid. As we scan an original image, the color of each pixel is processed as follows. If the color falls within the spatial extent of an existing leaf node, then add it to the node's color accumulator and increase the node's pixel count by 1. Otherwise, use the color to initialize the color accumulator of a new leaf node and set the node's pixel counter to 1. If this increases the number of leaf nodes to $k + 1$, merge some of the existing leaf nodes (leaves with greatest depth first) into their parent node, which becomes a new leaf node whose color accumulator takes on the sum of the accumulated color values from the children, and whose pixel counter gets the total of the children's pixel count.

Note that each splitting operation in median-cut or center-cut is performed either at the median or the midpoint of the longest dimension of a bounding box, and the merging operation in the octree algorithm is along predetermined spatial boundaries. This leaves the possibility of separating color points that are close to each other in a naturally forming cluster into different clusters.

Agglomerative clustering. There are other bottom-up approaches where we start with n clusters each of which contains one original color, and merge clusters without regard to preset spatial boundaries. In a method that relies on a three-dimensional representation of all 24-bit original colors as well as the clusters they belong to [35], we begin with n initial clusters that have the smallest bounding boxes. By gradually increasing the size limit on bounding boxes (with increments 2, 1, and 4 for red, green, and blue, respectively, to partially compensate for the nonuniform nature of the RGB color space), we merge neighboring clusters into larger ones to reduce the number of clusters. For a given size limit, we search the vicinity of each existing cluster S in the three-dimensional data structure to find candidates to merge, that is, clusters that can fit into a new bounding box for S that satisfies the size limit. The process terminates when k clusters remain.

In addition to limiting the size of bounding boxes, the criterion for merging clusters may also be based on variance or distance between centroids (see below).

Variance-based methods. The two-step top-down or bottom-up methods we have discussed so far decouple the formation of clusters and the computation of a representative for each cluster (the centroid) in the sense that the two steps are designed to achieve different numerical objectives: evenly distributed pixel population or size-restricted bounding boxes for clustering, and minimum variance for selecting cluster representatives after clustering. Several approximation algorithms are devised with variance-based criteria for the clustering step as well.

A K-means algorithm starts with an initial selection of quantized colors $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_k$, which may simply be evenly spaced points in the color space, or the result of some other algorithm. It then partitions the n original colors into k clusters $S_1, S_2, \ldots, S_k$ such that $\boldsymbol{c} \in S_i$ if $d(\boldsymbol{c}, \boldsymbol{q}_i) \leq d(\boldsymbol{c}, \boldsymbol{q}_j)$ for all j, $1 \leq j \leq k$. After the partition it calculates the centroid of each cluster S_i as the cluster's new representative $\boldsymbol{q}_i'$. The algorithm terminates when the relative reduction in overall quantization error from the previous choice of quantized colors is below a preset threshold. This relative reduction may be defined as $\frac{E-E'}{E'}$, where previous overall quantization error $E = \sum_{1\leq i\leq k} \sum_{\boldsymbol{c}\in S_i} P(\boldsymbol{c})d^2(\boldsymbol{c}, \boldsymbol{q}_i)$ and current overall quantization error $E' = \sum_{1\leq i\leq k} \sum_{\boldsymbol{c}\in S_i} P(\boldsymbol{c})d^2(\boldsymbol{c}, \boldsymbol{q}_i')$. Otherwise, the algorithm reiterates the partitioning step (followed by the recalculation of centroids) using the newly selected quantized colors. This quantization method is rather time consuming (with $O(nk)$ for each iteration), and its convergence at best leads to a locally optimal solution that is influenced by the initial selection of quantized colors [36–38].

In one of the bottom-up approaches we merge clusters under the notion of pairwise nearest neighbors [39]. Each iteration of the algorithm entails searching among current clusters to find two candidates, viz., S_i and S_j that are the closest neighbors, that is, two that when merged together into $S_{ij} = S_i \cup S_j$, will result in minimum sum of squared errors for S_{ij}: $\sum_{\boldsymbol{c}\in S_{ij}} P(\boldsymbol{c})d^2(\boldsymbol{c}, \boldsymbol{\mu}_{ij})$, where $\boldsymbol{\mu}_{ij}$ is the centroid of S_{ij}. A full implementation of this method is rather time-consuming since it would take at least $O(n\log\ n)$ just for the first iteration. To this end a k–d tree, where existing clusters (each cluster is spatially located at its centroid) are grouped into buckets (roughly equal number of clusters in each buckets), is used to restrict the search for pairwise nearest neighbors within each bucket (one pair per bucket). The pair that will result in the lowest sum of squared errors is merged first, then the pair in another bucket that yields the second lowest error sum, etc. The tree is rebalanced to account for the merged clusters when a certain percentage (e.g., 50%) of the identified pairs have been merged.

Another bottom-up method [40] randomly samples the input image for original colors and their frequencies; sorts the list of sampled original colors based on their frequencies in ascending order; and merges each color $\boldsymbol{c}_i$, starting from the top of the list (i.e., low frequency first), with its nearest neighbor $\boldsymbol{c}_j$, chosen based on a weighted squared Euclidean distance $\frac{P(\boldsymbol{c}_i)P(\boldsymbol{c}_j)}{P(\boldsymbol{c}_i)+P(\boldsymbol{c}_j)}d^2(\boldsymbol{c}_i, \boldsymbol{c}_j)$ to favor the merging of pairs of low-frequency colors. Each pair of merged colors is removed from the current list and replaced by $\boldsymbol{c}_{ij} = \frac{P(\boldsymbol{c}_i)\boldsymbol{c}_i + P(\boldsymbol{c}_j)\boldsymbol{c}_j}{P(\boldsymbol{c}_i)+P(\boldsymbol{c}_j)}$, with $P(\boldsymbol{c}_{ij}) = P(\boldsymbol{c}_i) + P(\boldsymbol{c}_j)$, which will be handled as an ordinary color during the next iteration of sorting and pairwise merging. The algorithm terminates when k colors remain on the list, which are used as quantized colors.

In a couple of approaches that follow the strategy of hierarchical subdivision, we start with a single cluster containing all original colors, and repeatedly partition the cluster S whose sum of squared errors $\sum_{\boldsymbol{c}\in S} P(\boldsymbol{c})d^2(\boldsymbol{c}, \boldsymbol{\mu})$, also termed weighted variance, with $\boldsymbol{\mu}$ being the centroid of S, is the highest. We move an orthogonal cutting plane along each of the three dimensions of the RGB color cube to search for a position that divides the chosen cluster into two. One way to determine the orientation and position of the cutting plane is to project color points in the cluster, bounded by $r_1 \leq r \leq r_2$, $g_1 \leq g \leq g_2$, and $b_1 \leq b \leq b_2$, onto each of the three color axis; find the threshold that minimizes the weighted sum of projected variances of the two intervals adjoining at the threshold for each axis; and run the cutting plane perpendicular to and through the threshold on the axis that gives the minimum sum of projected variances [41]. More specifically, the frequency of a projected point on the r-axis is $P(r, 0, 0) = \sum_{g_1\leq g\leq g_2} \sum_{b_1\leq b\leq b_2} P(r, g, b)$. Likewise, we have $P(0, g, 0) = \sum_{r_1\leq r\leq r_2} \sum_{b_1\leq b\leq b_2} P(r, g, b)$ for the g-axis and $P(0, 0, b) = \sum_{r_1\leq r\leq r_2} \sum_{g_1\leq g\leq g_2} P(r, g, b)$ for the b-axis. Given an axis along with a series of projected points between l and m, a threshold $l < t \leq m$ partitions the points into two intervals $[l, t-1]$

and $[t, m]$, with the resulting weighted sum of projected variances being $E_t = \sum_{l\le i\le t-1} P_i(i-\mu_1)^2 + \sum_{t\le i\le m} P_i(i-\mu_2)^2$, where μ_1 and μ_2 are the means of the two intervals, respectively, and $P_i = P(i, 0, 0)$, $P(0, i, 0)$, or$P(0, 0, i)$. The optimal threshold value that minimizes E_t is in the range of $[\frac{l+\mu}{2}, \frac{\mu+m}{2}]$ and maximizes $\frac{w_1}{w_2}(\mu-\mu_1)^2$, where μ is the mean of the projected points in $[l, m]$, $w_1 = \sum_{l\le i\le t-1} P_i$, and $w_2 = \sum_{t\le i\le m} P_i$ are the weights for the two respective intervals [42].

Another way to determine the cutting plane is to minimize the sum of weighted variances (without projecting points onto the three color axes) on both sides of the plane [43]. A rectangular bounding box is now defined by $r_1 < r \le r_2$, $g_1 < g \le g_2$, and $b_1 < b \le b_2$; and it is denoted by $\Omega(\boldsymbol{c}_l, \boldsymbol{c}_m]$, where $\boldsymbol{c}_l = (r_1, g_1, b_1)$ and $\boldsymbol{c}_m = (r_2, g_2, b_2)$. And we define $M_d(\boldsymbol{c}_t) = \sum_{\boldsymbol{c}\in\Omega(o,\boldsymbol{c}_t]} \boldsymbol{c}^d P(\boldsymbol{c})$, with $d = 0, 1, 2$, $\boldsymbol{c}^0 = 1$, $\boldsymbol{c}^2 = \boldsymbol{c}\boldsymbol{c}^T$, and $\boldsymbol{o}$ being a reference point such that $\sum_{\boldsymbol{c}\in\Omega(-\infty,\boldsymbol{o}]} P(\boldsymbol{c}) = 0$. We precompute and store $M_d(\boldsymbol{c})$, $d = 0, 1, 2$, for each grid point in the RGB space to facilitate efficient computation of the pixel population $w(\boldsymbol{c}_l, \boldsymbol{c}_m]$, mean $\boldsymbol{\mu}(\boldsymbol{c}_l, \boldsymbol{c}_m]$, and weighted variance $E(\boldsymbol{c}_l, \boldsymbol{c}_m]$ of any cluster of image colors bounded by $\Omega(\boldsymbol{c}_l, \boldsymbol{c}_m]$:

$$w(\boldsymbol{c}_l, \boldsymbol{c}_m] = \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_m]} P(\boldsymbol{c})$$

$$\boldsymbol{\mu}(\boldsymbol{c}_l, \boldsymbol{c}_m] = \frac{\displaystyle\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_m]} \boldsymbol{c}P(\boldsymbol{c})}{w(\boldsymbol{c}_l, \boldsymbol{c}_m]}$$

$$E(\boldsymbol{c}_l, \boldsymbol{c}_m] = \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_m]} P(\boldsymbol{c})d^2(\boldsymbol{c}, \boldsymbol{\mu}(\boldsymbol{c}_l, \boldsymbol{c}_m]) = \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_m]} \boldsymbol{c}^2 P(\boldsymbol{c}) - \frac{\left(\displaystyle\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_m]} \boldsymbol{c}P(\boldsymbol{c})\right)^2}{w(\boldsymbol{c}_l, \boldsymbol{c}_m]}$$

The evaluation of these items in $O(1)$ time is made possible by designating the remaining six corners of the bounding box as

$$\begin{aligned} \boldsymbol{c}_a &= (r_2, g_1, b_1); & \boldsymbol{c}_b &= (r_1, g_2, b_1); & \boldsymbol{c}_c &= (r_1, g_1, b_2); \\ \boldsymbol{c}_d &= (r_1, g_2, b_2); & \boldsymbol{c}_e &= (r_2, g_1, b_2); & \boldsymbol{c}_f &= (r_2, g_2, b_1) \end{aligned}$$

and applying the rule of inclusion–exclusion to obtain

$$\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_m]} f(\boldsymbol{c})P(\boldsymbol{c}) = \left(\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{o},\boldsymbol{c}_m]} + \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{o},\boldsymbol{c}_a]} + \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{o},\boldsymbol{c}_b]} + \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{o},\boldsymbol{c}_c]} - \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{o},\boldsymbol{c}_d]} - \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{o},\boldsymbol{c}_e]} - \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{o},\boldsymbol{c}_f]} - \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{o},\boldsymbol{c}_l]}\right) f(\boldsymbol{c})P(\boldsymbol{c})$$

where $f(\boldsymbol{c})$ may be 1, $\boldsymbol{c}$, or $\boldsymbol{c}^2$. Furthermore, to determine a cutting plane for $\Omega(\boldsymbol{c}_l, \boldsymbol{c}_m]$, we need to minimize $E(\boldsymbol{c}_l, \boldsymbol{c}_t] + E(\boldsymbol{c}_t, \boldsymbol{c}_m]$, with $\boldsymbol{c}_t = (r, g_2, b_2)|_{r_1<r\le r_2}$ or $(r_2, g, b_2)|_{g_1<g\le g_2}$ or $(r_2, g_2, b)|_{b_1<b\le b_2}$. Since

$$E(\boldsymbol{c}_l, \boldsymbol{c}_t] + E(\boldsymbol{c}_t, \boldsymbol{c}_m] = \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_m]} \boldsymbol{c}^2 P(\boldsymbol{c}) - \frac{\left(\displaystyle\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_t]} \boldsymbol{c}P(\boldsymbol{c})\right)^2}{w(\boldsymbol{c}_l, \boldsymbol{c}_t]} - \frac{\left(\displaystyle\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_t,\boldsymbol{c}_m]} \boldsymbol{c}P(\boldsymbol{c})\right)^2}{w(\boldsymbol{c}_t, \boldsymbol{c}_m]}$$

minimizing $E(\boldsymbol{c}_l, \boldsymbol{c}_t] + E(\boldsymbol{c}_t, \boldsymbol{c}_m]$ is equivalent to maximizing

$$\frac{\left(\displaystyle\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_t]} \boldsymbol{c}P(\boldsymbol{c})\right)^2}{w(\boldsymbol{c}_l, \boldsymbol{c}_t]} + \frac{\left(\displaystyle\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_t,\boldsymbol{c}_m]} \boldsymbol{c}P(\boldsymbol{c})\right)^2}{w(\boldsymbol{c}_t, \boldsymbol{c}_m]} = \frac{\left(\displaystyle\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_t]} \boldsymbol{c}P(\boldsymbol{c})\right)^2}{w(\boldsymbol{c}_l, \boldsymbol{c}_t]} + \frac{\left(\displaystyle\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_m]} \boldsymbol{c}P(\boldsymbol{c}) - \sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_t]} \boldsymbol{c}P(\boldsymbol{c})\right)^2}{w(\boldsymbol{c}_l, \boldsymbol{c}_m] - w(\boldsymbol{c}_l, \boldsymbol{c}_t]}$$

where $w(\boldsymbol{c}_l, \boldsymbol{c}_m]$ and $\sum_{\boldsymbol{c}\in\Omega(\boldsymbol{c}_l,\boldsymbol{c}_m]} \boldsymbol{c}P(\boldsymbol{c})$ are the constants.

In addition to exploring cutting planes that are perpendicular to axes of the color coordinate system, we may also look into other orientations as well. For example, we may take a random sample of the cluster S that has the highest weighted variance, and subdivide it into two in the following way [44]. For every linearly separable two-clustering of the sample set T into T_1 and T_2, compute their centroids t_1 and t_2, divide S by the perpendicular bisector of $\overline{t_1 t_2}$, then compute the centroids s_1 and s_2 of the two resulting subsets, and divide S again by the perpendicular bisector of $\overline{s_1 s_2}$. Finally, choose among all second bisectors the one that yields the minimum sum of weighted variances of the two subsets of S to divide S.

Alternatively, we may place cutting planes orthogonally to each cluster's principal axis, which is found from the largest eigenvalue and the corresponding principal eigenvector of the cluster's covariance matrix [45]. During each iteration of the subdivision process, the algorithm finds a cluster whose principal eigenvalue is the highest among all existing clusters, and bisects the found cluster with a plane that is perpendicular to the corresponding eigenvector and through the cluster's mean. The two resulting clusters become independent candidates for further partitioning in subsequent iterations.

Minimizing total quantization errors. Principal analysis is also the basis for an approximation method for the minimization of total quantization errors [46]. The algorithm is inspired by the observation that colors in any given image tend to form a cluster that spreads out more in terms of differences in luminance than in chromaticity variations. It first finds the principal axis for the entire set of original colors. It then introduces parallel cutting planes that are perpendicular to the principal axis to minimize $\sum_{1\le i\le \kappa} \sum_{\boldsymbol{c}\in S_i} P(\boldsymbol{c})d(\boldsymbol{c}, \boldsymbol{\mu}_i)$, where $\boldsymbol{\mu}_i$ is the centroid of S_i, and κ is increased as cutting planes are introduced one by one by way of dynamic programming until none of the resulting clusters has a strongly biased orientation in the principal direction of the original set. Now if $\kappa = k$ the algorithm terminates with k clusters; otherwise, it continues to subdivide the κ existing clusters, either by using one of the hierarchical methods, or by splitting each chosen cluster with a cutting plane that is perpendicular to the cluster's principal axis and minimizes the total quantization errors of the two resulting subsets.

Minimizing maximum intercluster distance. Unique among quantization algorithms that are based on numerical criteria, and unlike hierarchical methods that partition or merge clusters based on local information (i.e., color points inside a restricted spatial extent), the following method achieves proven tight approximation to global optimality for its clustering operation. The method attempts to minimize the maximum quantization error across the entire image by partitioning original colors into tight clusters under a formal notion of minimizing the maximum intercluster[2] distance: finding a partition of n points in an m-dimensional Euclidean space into k disjoint clusters $S_1, S_2, \ldots, S_k$ such that $\max(M_1, M_2, \ldots, M_k)$, where M_i is the maximum distance between two points in cluster S_i, is minimized [47]. This minimization problem is polynomial solvable for $m = 1$ [23] and NP-hard for $m = 2$ [48]. When $m = 3$, which is typically the case in color image quantization, even finding a partition with maximum intercluster distance less than two times the optimal solution value, referred to as the $(2 - \varepsilon)$-approximation problem, is NP-hard for all $\varepsilon > 0$ [48,49]. Hence we make use of an efficient 2-approximation algorithm that has worst-case time complexity $O(nk)$ [48]:

```
$S_1 = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n\}$;        // Start with a single cluster containing all original colors
$\boldsymbol{h}_1 = \boldsymbol{c}_1$;        // Each cluster has a designated point as the head of the cluster
for ($x = 1$; $x < k$; $x$++) {
    $d = \max\{d(\boldsymbol{c}_i, \boldsymbol{h}_j) \mid \boldsymbol{c}_i \in S_j, 1 \le i \le n$, and $1 \le j \le x\}$;
```

[2] Since we are trying to minimize the maximum distance between color points in each cluster it might be more appropriate to use the word intracluster. However, if we view each color point as a singleton cluster we are indeed minimizing the maximum intercluster distance. We adopt this second view to be consistent with the existing literature on clustering.

```
        $\boldsymbol{c}$ = one of the points whose distance to its respective cluster head is $d$;
        move $\boldsymbol{c}$ to $S_{x+1}$;
        $\boldsymbol{h}_{x+1} = \boldsymbol{c}$;
        for each $\boldsymbol{c}' \in (S_1 \cup S_2 \cup \ldots \cup S_x)$ {
            let $j$ be such that $\boldsymbol{c}' \in S_j$;
            if $(d(\boldsymbol{c}', \boldsymbol{h}_j) \geq d(\boldsymbol{c}', \boldsymbol{c}))$ move $\boldsymbol{c}'$ from $S_j$ to $S_{x+1}$;
        }
    }
```

86.5 Image-Dependent, Context-Sensitive Quantization

Context-sensitive quantization methods work with not only original colors and their frequencies, but also the image's context. The latter spans from the adjacency relationship between pixels, to the primitive elements of visual information above the pixel level (e.g., edges and boundaries), and to the overall meaning of the visual information that the image and various parts of the image convey. Effective use of contextual information should bring about a better balance for the allocation of resources (i.e., selection of quantized colors along with the mapping of colors) in terms of moderating visually offensive distortion than what we may achieve with context-free quantization.

86.5.1 Dithered Quantization

This color quantization method takes into consideration the impact of neighboring pixels on the viewer's perception of each individual pixel in both the original and the quantized images [50]. Let $\boldsymbol{c}_{x,y}$ be the color of pixel (x, y) in the original image, and $\boldsymbol{c}'_{x,y}$ the perceived color at (x, y), calculated by a linear blurring operation that convolutes the original image with a localized kernel to account for the phenomenon of spatial averaging in human vision (e.g., a Gaussian kernel of identical standard deviation for all color components, with choice of neighborhood size from 3×3 to 11×11). Similarly, let $\boldsymbol{q}_{x,y}$ be the color of pixel (x, y) in the quantized image, and $\boldsymbol{q}'_{x,y}$ the perceived color at (x, y), calculated by convoluting the quantized image with the same kernel. The goal of color quantization is now defined as the minimization of $\sum_{1\leq x\leq w} \sum_{1\leq y\leq h} d^2(\boldsymbol{c}'_{x,y}, \boldsymbol{q}'_{x,y})$, where w and h are the width and height of the image. Hence the task of quantization becomes finding the right set Q of quantized colors along with a proper assignment A of each pixel to a quantized color—simultaneous quantization and dithering.

A twofold minimization scheme—first optimize A for a fixed Q and then optimize Q for a fixed A—is iterated to achieve convergence to a local minimum of the stated goal. The optimization of A is solved by a local Iterative Conditional Mode (ICM) algorithm in Ref. [50], whereas more accurate results can be produced with deterministic annealing [51].

86.5.2 Feedback-Based Quantization

Short of being able to reliably predict the type, severity, and location of eye-catching artifacts in the quantized image, we may try to develop techniques to detect the artifacts, and use the findings as feedback to modify the behavior of the quantizer to alleviate the distortion. A couple of studies have looked into a commonly occurring type of visible artifacts, viz., the appearance of false contours in areas that have gradual shadings before quantization. These false contours are the direct result of mapping a series of smoothly changing colors into a low number of quantized colors that make up a staircase-like profile across a troubled area—the effect of simultaneous contrast can make the steps look more profound than they really are.

In an extension to an aforementioned variance-based quantization method [45], a weighting mechanism is activated after the number of clusters have reached a preset threshold (e.g., $\frac{2}{3}k$) to adjust the ranking of existing clusters for further subdivision. Each candidate cluster is now given a weight that represents the

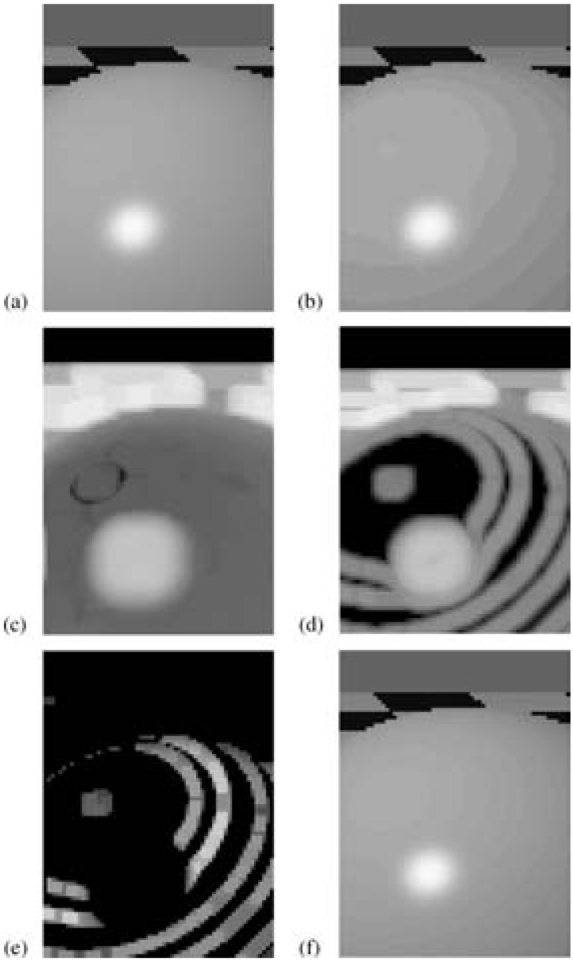


FIGURE 86.6 Detecting and reducing false contours.

size (number of interior pixels) of a continuous region that is colored by the cluster's representative in the quantized image. The candidate cluster whose principal eigenvalue multiplied by its weight is the highest is chosen for splitting. Doing so helps to eliminate large and uniformly colored regions. However, these regions are only necessary, rather than sufficient conditions for false contours. Even when they do border false contours, we need more than their sizes to distinguish a severe false contour from a minor one.

Another investigation involves an iterative process where findings from the last complete quantized image are used to requantize the original for better results [52,53]. An agglomerative clustering quantizer [35] that operates on 24-bit RGB colors is adapted to be the embedded quantization mechanism, since the usual 3-3-3 or 3-2-4 bit-cutting color-reduction technique itself causes false contours to appear when the original is a computer-synthesized or high-quality photographic image that depicts smooth/glossy surfaces.[3]

The system detects false contours in the quantized image by convoluting both the original (Figure 86.6[a]) and the quantized (Figure 86.6[b][4]) images with a set of 5 × 5 directional edge detectors. The detectors are

[3] The introduction of a random perturbation to slightly degrade the original before quantization tends to inhibit the occurrence of false contours in the quantized image.

[4] The effect of simultaneous contrast is visible when the image is reproduced with good fidelity: Each uniformly shaded band on the spherical surface in the foreground looks nonuniform—darker on the side that is adjacent to a brighter band and brighter on the side that is adjacent to a darker band.

first applied independently to each primary component. A magnitude value in Euclidean color distance is then calculated for each detector (the corresponding results in the red, green, and blue directions are first scaled by 2, 4, and 1, respectively, to make the measurement more indicative of the change in luminance), and the highest magnitude is recorded as the magnitude of the edge element in an edge map. Next, we construct a mask (Figure 86.6[c]) based on the edges in the original image's edge map, and use the mask to suppress their counterparts (i.e., the true edges) in the quantized image's edge map (Figure 86.6[d]). After further elimination of relatively insignificant edge elements by thresholding, the resulting edge map for the quantized image becomes a false contour map (Figure 86.6[e]), which identifies areas in the quantized image that border false contours. Each of these areas in turn identifies a set of original colors that have been mapped to the same quantized color, and need to be better preserved during requantization to alleviate the artifacts (Figure 86.6[f]—no error diffusion). The latter is accomplished by increasing the importance factor (initially all original colors have equal importance) of each affected original color, and having the quantizer restrict the growth of each cluster based on the highest importance factor of its constituents. The increment in importance for colors in an area that needs to reduce quantization errors is proportional to

$$maxedge^2 \times (1 + tcd/maxtcd) \times (1 + pp/maxpp)$$

where *maxedge* is the magnitude of the highest edge element from the corresponding region of the false contour map, *tcd* the total discrepancies between the quantized color and the colors in the area, *pp* the area's pixel population, and *maxtcd* and *maxpp* are the maximum *tcd* and *pp*, respectively, of all areas in the image that are identified by the false contour detection process.

Figure 86.7 shows a gray-scale reproduction of a ray-traced image with photo-edited background (blue sky, white cloud, and brown mountain), where the five model cars are in silver, green, gold, orange, and cyan, respectively. Figure 86.8 is the gray-scale reproduction of the result of quantizing Figure 86.7 to 256

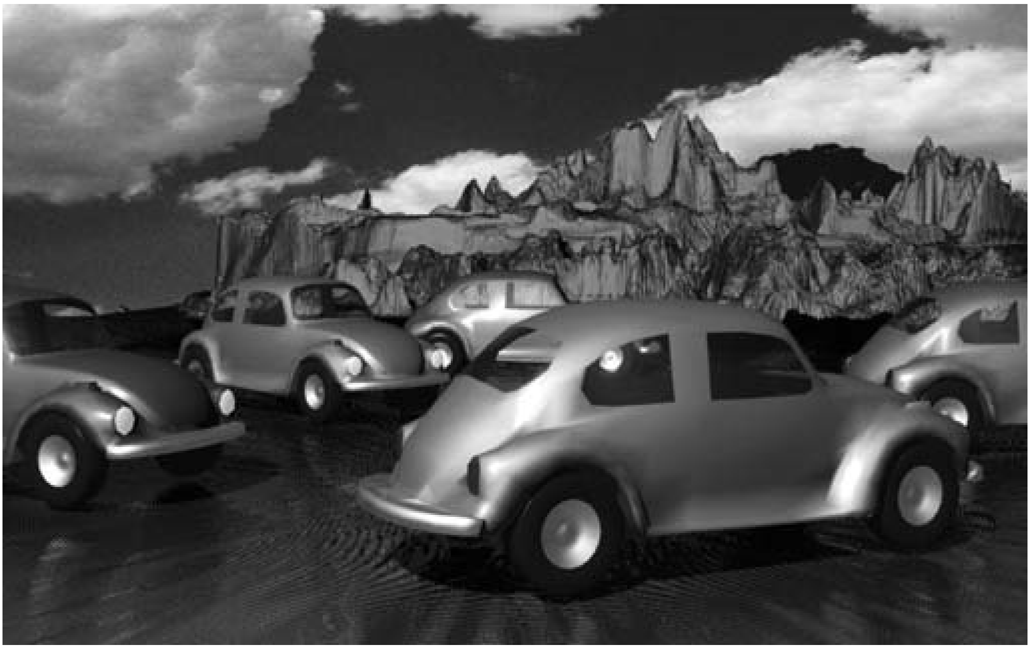


FIGURE 86.7 Gray-scale reproduction of a ray-traced color image with added background.

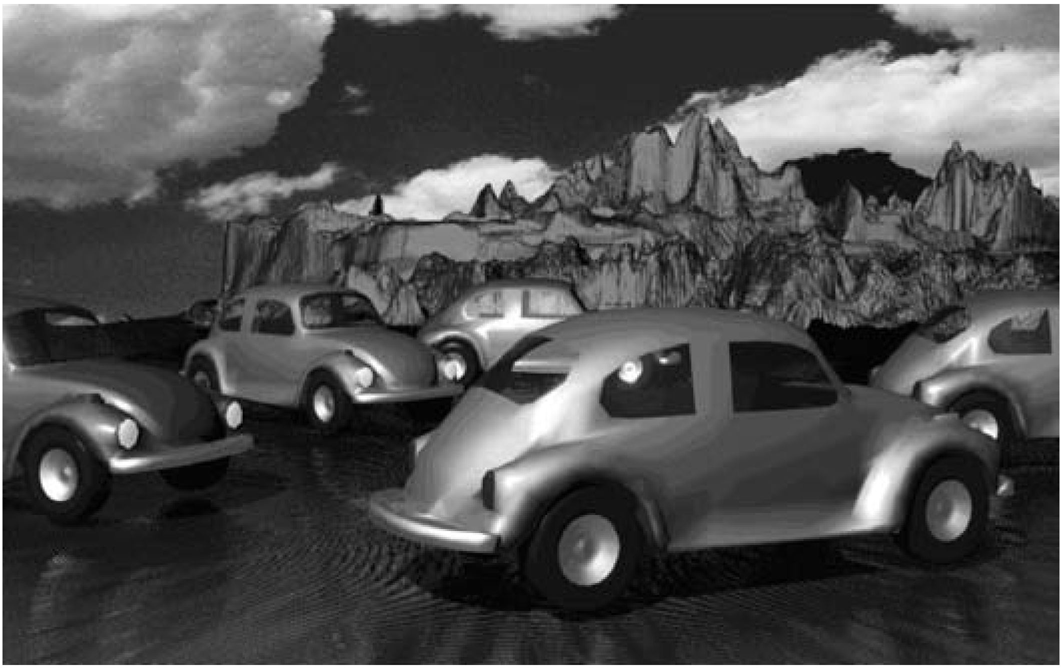


FIGURE 86.8 Context-free quantization.

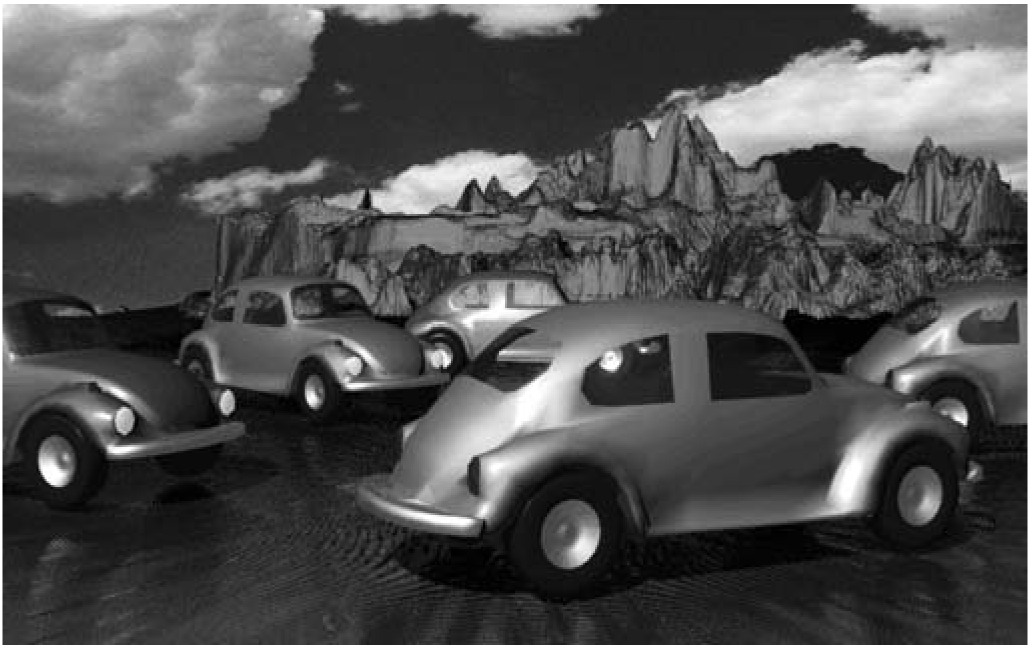


FIGURE 86.9 Context-sensitive quantization.

colors using the original agglomerative quantizer, where false contours are clearly visible on four of the cars. Figure 86.9 is the gray-scale reproduction of what is produced by the feedback-based system at the end of the 47th iteration (without error diffusion), where false contours are greatly reduced with no clear degradation elsewhere.

References

[1] Lindbloom, B. J., Accurate color reproduction for computer graphics applications, *Comput. Graphics*, 23(3), 117, 1989.

[2] Stone, M. C., Cowan, W. B., and Beatty, J. C., Color gamut mapping and the printing of digital color images, *ACM Trans. Graphics*, 7(4), 249, 1988.

[3] Gentile, R. S., Allebach, J. P., and Walowit, E., Quantization of color images based on uniform color spaces, *J. Imaging Technol.*, 16(1), 11, 1990.

[4] Kurz, B. J., Optimal color quantization for color displays, in *IEEE Proc. Computer Vision and Pattern Recognition*, 1983, p. 217.

[5] Itten, J., *The Elements of Color*, Wiley, New York, 2003.

[6] Floyd, R. and Steinberg, L., An adaptive algorithm for spatial gray scale, *Int. Symp. Digest of Technical Papers*, Society for Information Display, 1975, p. 36.

[7] Knuth, D. E., Digital halftones by dot diffusion, *ACM Trans. Graphics*, 6, 245, 1987.

[8] Ostromoukhov, V., A simple and efficient error-diffusion algorithm, in *Proc. Conf. on Computer Graphics and Interactive Techniques*, 2001, p. 567.

[9] Zhang, Y. and Webber, R. E., Space diffusion: an improved parallel halftoning technique using space-filling curves, in *Proc. Conf. on Computer Graphics and Interactive Techniques*, 1993, p. 305.

[10] Wyszecki, G. and Stiles, W. S., *Color Science: Concepts and Methods, Quantitative Data and Formulae*, 2nd ed., Wiley, New York, 1982.

[11] Hill, B., Roger, Th., and Vorhagen, F. W., Comparative analysis of the quantization of color spaces on the basis of the CIELAB color-difference formula, *ACM Trans. Graphics*, 16(2), 109, 1997.

[12] Kasson, J. M. and Plouffe, W., An analysis of selected computer interchange color spaces, *ACM Trans. Graphics*, 11(4), 373, 1992.

[13] Weinman, L. and Heavin, B., *Coloring Web Graphics*, 2nd ed., New Riders, Indianapolis, IN, 1997.

[14] Forney, G. D., Jr., The Viterbi algorithm, *Proc. IEEE*, 61, 169, 1984.

[15] Viterbi, A. J., Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Trans. Inf. Theor.*, 13, 260, 1967.

[16] Cheng, S. S., Xiong, Z., and Wu, X., Fast trellis-coded color quantization of images, *Real-Time Imaging*, 8, 265, 2002.

[17] Marcellin, M. W. and Fischer, T. R., Trellis coded quantization of memoryless and Gaussian-Markov sources, *IEEE Trans. Commun.*, 38, 82, 1990.

[18] Ungerboeck, G., Channel coding with multilevel/phase signals, *IEEE Trans. Inf. Theor.*, 28, 55, 1982.

[19] Mojsilović, A. and Soljanin, E., Color quantization and processing by Fibonacci lattices, *IEEE Trans. Image Process.*, 10(11), 1712, 2001.

[20] Jain, A. K., Murty, M. N., and Flynn, P. J., Data clustering: a review, *ACM Comput. Surv.*, 31(3), 264, 1999.

[21] Lloyd, S. P., Least squares quantization in PCM, Unpublished Bell Laboratories memorandum, 1957; also *IEEE Trans. Inf. Theor.*, IT-28, 129, 1982.

[22] Max, J., Quantizing for minimum distortion, *IRE Trans. Inf. Theor.*, IT-6, 7, 1960.

[23] Brucker, P., On the complexity of clustering problems, in *Optimization and Operations Research*, Henn, R., Korte, B., and Oettli, W., Eds., Springer, Berlin, 1978, p. 45.

[24] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979, Problem MS9.

[25] Garey, M. R., Johnson, D. S., and Witsenhausen, H. S., The complexity of the generalized Lloyd-Max problem, *IEEE Trans. Inf. Theor.*, IT-28(2), 255, 1982.

[26] Megiddo, N. and Supowit, K. J., On the complexity of some common geometric location problems, *SIAM J. Comput.*, 13, 182, 1984.

[27] Heckbert, P., Color image quantization for frame buffer display, *Comput. Graphics*, 16(3), 297, 1982.

[28] Braudaway, G. W., A procedure for optimum choice of a small number of colors from a large color palette for color imaging, *Proc. Electron. Imaging*, 71, 1987.

[29] Kim, N. and Kehtarnavaz, N., DWT-based scene-adaptive color quantization, *Real-Time Imaging*, 11 (5–6), 443, 2005.

[30] Lehar, A. F. and Stevens, R. J., High-speed manipulation of the color chromaticity of digital images, *IEEE Comput. Graphics Appl.*, 4, 34, 1984.

[31] Stevens, R. J., Lehar, A. F., and Preston, F. H., Manipulation and presentation of multidimensional image data using the Peano scan, *IEEE Trans. Pattern Anal. Mach. Intell.*, 5, 520, 1983.

[32] Joy, G. and Xiang, Z., Center-cut for color-image quantization, *Visual Comput.*, 10(1), 62, 1993.

[33] Samet, H., *Applications of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

[34] Gervautz, M. and Purgathofer, W., A simple method for color quantization: octree quantization, in *New Trends in Computer Graphics*, Magnenat-Thalmann, N. and Thalmann, D., Eds., Springer, Berlin, 1988, p. 219.

[35] Xiang, Z. and Joy, G., Color image quantization by agglomerative clustering, *IEEE Comput. Graphics Appl.*, 14(3), 44, 1994.

[36] Gray, R. M., Kieffer, J. C., and Linde, Y., Locally optimal block quantizer design, *Inf. Contr.*, 45, 178, 1980.

[37] Linde, Y., Buzo, A., and Gray, R. M., An algorithm for vector quantizer design, *IEEE Trans.Commun.*, 28(1), 84, 1980.

[38] Selim, S. Z. and Ismail, M. A., K-means-type algorithms: a generalization convergence theorem and characterization of local optimality, *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-6(1), 81, 1984.

[39] Equitz, W. H., A new vector quantization clustering algorithm, *IEEE Trans. Acoustics, Speech, Signal Process.*, 37(10), 1568, 1989.

[40] Dixit, S. S., Quantization of color images for display/printing on limited color output devices, *Comput. Graphics*, 15(4), 561, 1991.

[41] Wan, S. J., Prusinkiewicz, P., and Wong, S. K. M., Variance-based color image quantization for frame buffer display, *Color Res. Appl.*, 15(1), 52, 1990.

[42] Wong, S. K. M., Wan, S. J., and Prusinkiewicz, P., Monochrome image quantization, *Proc. Canadian Conf. Electrical and Computer Engineering*, 1989, p. 28.

[43] Wu, X., Efficient statistical computations for optimal color quantization, in *Graphics Gems II*, Arvo, J., Ed., Academic Press, New York, 1991, p. 126.

[44] Inaba, M, Imai, H., Nakade, M., and Sekiguchi, T., Application of an effective geometric clustering method to the color quantization problem, *Proc. Symp. on Computational Geometry*, 1997, p. 477.

[45] Orchard, M. T. and Bouman, C. A., Color quantization of images, *IEEE Trans. Signal Process.*, 39(12), 2677, 1991.

[46] Wu, X., Color quantization by dynamic programming and principle analysis, *ACM Trans. Graphics*, 11(4), 348, 1992.

[47] Xiang, Z., Color image quantization by minimizing the maximum intercluster distance, *ACM Trans. Graphics*, 16(3), 260, 1997.

[48] Gonzalez, T. F., Clustering to minimize the maximum intercluster distance, *Theor. Comput. Sci.*, 38(2–3), 293, 1985.

[49] Sahni, S. and Gonzalez, T., P-complete approximation problems, *JACM*, 23(3), 555, 1976.

[50] Buhmann, J. M., Fellner, D. W., Held, M., Ketterer, J., and Puzicha, J., Dithered color quantization, *Comput. Graphics Forum*, 17(3), 219, 1998.

[51] Ketterer, J., Puzicha, J., Held, M., Fischer, M., Buhmann, J. M., and Fellner, D., On spatial quantization of color images, *Proc. European Conf. on Computer Vision*, 1998, p. 563.

[52] Joy, G. and Xiang, Z., Reducing false contours in quantized color images,*Comput. Graphics*, 20(2), 231, 1996.

[53] Xiang, Z. and Joy, G., Feedback-based quantization of color images, *Proc. SPIE 2182: Image and Video Processing II*, 1994, p. 34.

[54] Eschbach, R. and Knox, K. T., Error-diffusion algorithm with edge enhancement, *J. Opt. Soc. Am. A*, 8(12), 1844, 1991.

Index

Computer Science/Computer Engineering/Computing

CHAPMAN & HALL/CRC COMPUTER and INFORMATION SCIENCE SERIES

Handbook of Approximation Algorithms and Metaheuristics

Delineating the tremendous growth in this area, the **Handbook of Approximation Algorithms and Metaheuristics** covers fundamental, theoretical topics as well as advanced, practical applications. It is the first book to comprehensively study both approximation algorithms and metaheuristics.

Starting with basic approaches, the handbook presents the methodologies to design and analyze efficient approximation algorithms for a large class of problems, and to establish inapproximability results for another class of problems. It also discusses local search, neural networks, and metaheuristics, as well as multiobjective problems, sensitivity analysis, and stability. After laying this foundation, the book applies the methodologies to classical problems in combinatorial optimization, computational geometry, and graph problems. In addition, it explores large-scale and emerging applications in networks, bioinformatics, VLSI, game theory, and data analysis.

Undoubtedly sparking further developments in the field, this handbook provides the essential techniques to apply approximation algorithms and metaheuristics to a wide range of problems in computer science, operations research, computer engineering, and economics. Armed with this information, researchers can design and analyze efficient algorithms to generate near-optimal solutions for a wide range of computational intractable problems.

Features

- Describes basic methodologies that include restriction, greedy, relaxation, rounding, primal-dual, local search, transformation, and metaheuristics
- Explains polynomial time and fully polynomial time approximation schemes, including standard, asymptotic, and randomized
- Reviews approximation algorithms for bin packing, the traveling salesperson problem, and Steiner trees
- Discusses computational geometry and graph applications, such as triangulations, pair decompositions, partitioning, maximum planar subgraphs, edge disjoint paths and unsplittable flow, and communication spanning trees
- Presents the latest algorithmic applications, including wireless ad hoc networks, microarray analysis, and global routing

C5505

ISBN 1-58488-550-5

90000

9 781584 885504

Chapman & Hall/CRC
Taylor & Francis Group
an informa business
www.taylorandfrancisgroup.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
270 Madison Avenue
New York, NY 10016
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK

www.crcpress.com